# 王道考研——组成原理

WWW.CSKAOYAN.COM

## 第五章 中央处理器

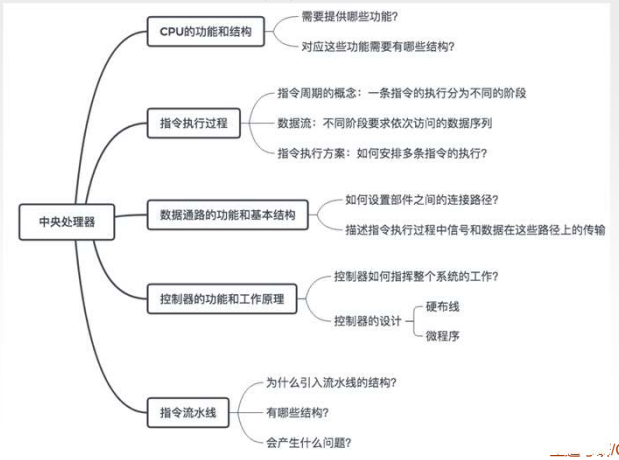
本节内容

中央处理器

CPU的功能和基本结构

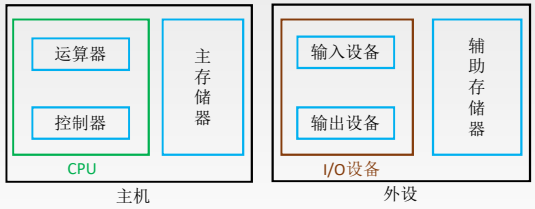


本章总览






CPU的功能



## CPU的功能


1. **指令控制**。完成取指令、分析指令和执行指令的操作，即程序的顺序控制。

2. **操作控制**。一条指令的功能往往是由若干操作信号的组合来实现的。CPU管理并产生由内存取出的每条指令的操作信号，把各种操作信号送往相应的部件，从而控制这些部件按指令的要求进行动作。

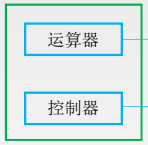
3. **时间控制**。对各种操作加以时间上的控制。时间控制要为每条指令按时间顺序提供应有的控制信号。

4. **数据加工**。对数据进行算术和逻辑运算。

5. **中断处理**。对计算机运行过程中出现的异常情况和特殊请求进行处理。



## 运算器和控制器的功能




对数据进行加工

协调并控制计算机各部件执行程序的指令序列，
基本功能包括取指令、分析指令、执行指令

取指令：自动形成指令地址；自动发出取指令的命令。

分析指令：操作码译码(分析本条指令要完成什么操作)；
产生操作数的有效地址。

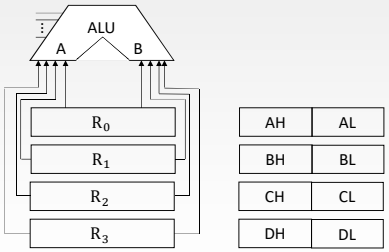
执行指令：根据分析指令得到的“操作命令”和“操作数地址”，
形成操作信号控制序列，控制运算器、存储器以及I/O
设备完成相应的操作。

中断处理：管理总线及输入输出；处理异常情况(如掉电)和特殊请求(如打印机请求打印一行字符)。



## 运算器的基本结构

1. 算术逻辑单元：主要功能是进行算术/逻辑运算。
2. 通用寄存器组：如AX、BX、CX、DX、SP等，用于存放操作数（包括源操作数、目的操作数及中间结果）和各种地址信息等。SP是堆栈指针，用于指示栈顶的地址。



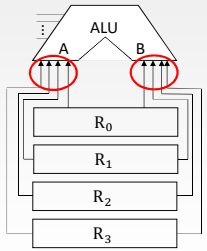

| AH | AL |
|---|---|
| BH | BL |
| CH | CL |
| DH | DL |
| SP | |

专用数据通路方式：根据指令执行过程中的数据和地址的流动方向安排连接线路。



## 运算器的基本结构

1. 算术逻辑单元：主要功能是进行算术/逻辑运算。
2. 通用寄存器组：如AX、BX、CX、DX、SP等，用于存放操作数（包括源操作数、目的操作数及中间结果）和各种地址信息等。SP是堆栈指针，用于指示栈顶的地址。



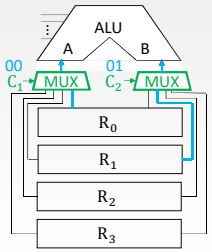

如果直接用导线连接，相当于多个寄存器同时并且一直向ALU传输数据

解决方法1. 使用多路选择器

专用数据通路方式：根据指令执行过程中的数据和地址的流动方向安排连接线路。

## 运算器的基本结构

1. 算术逻辑单元：主要功能是进行算术/逻辑运算。
2. 通用寄存器组：如AX、BX、CX、DX、SP等，用于存放操作数（包括源操作数、目的操作数及中间结果）和各种地址信息等。SP是堆栈指针，用于指示栈顶的地址。




如果直接用导线连接，相当于多个寄存器同时并且一直向ALU传输数据

解决方法1. 使用多路选择器

根据控制信号选择一路输出

解决方法2. 使用三态门

可以控制每一路是否输出

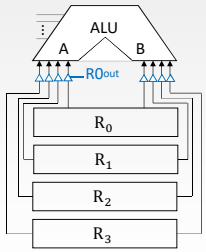
专用数据通路方式：根据指令执行过程中的数据和地址的流动方向安排连接线路。



## 运算器的基本结构

CPU内部单总线方式：将所有寄存器的输入端和输出端都连接到一条公共的通路上。

1. 算术逻辑单元：主要功能是进行算术/逻辑运算。
2. 通用寄存器组：如AX、BX、CX、DX、SP等，用于存放操作数（包括源操作数、目的操作数及中间结果）和各种地址信息等。SP是堆栈指针，用于指示栈顶的地址。




如果直接用导线连接，相当于多个寄存器同时并且一直向ALU传输数据

解决方法1. 使用多路选择器

根据控制信号选择一路输出

解决方法2. 使用三态门

可以控制每一路是否输出

如：ROout为1时$R_0$中的数据输出到A端，

ROout为0时$R_0$中的数据无法输出到B端

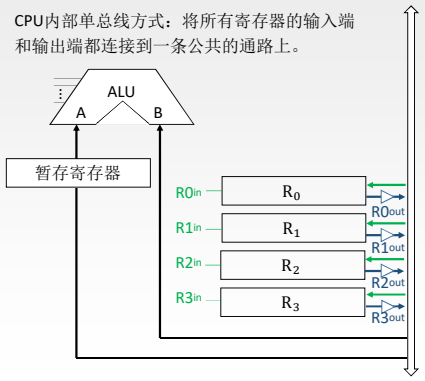
性能较高，基本不存在数据冲突现象，但结构复杂，硬件量大，不易实现。

专用数据通路方式：根据指令执行过程中的数据和地址的流动方向安排连接线路。



## 运算器的基本结构

CPU内部单总线方式：将所有寄存器的输入端和输出端都连接到一条公共的通路上。



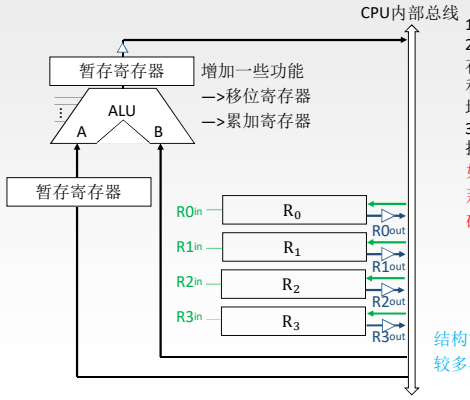

1. 算术逻辑单元：主要功能是进行算术/逻辑运算。
2. 通用寄存器组：如AX、BX、CX、DX、SP等，用于存放操作数（包括源操作数、目的操作数及中间结果）和各种地址信息等。SP是堆栈指针，用于指示栈顶的地址。
3. 暂存寄存器：用于暂存从主存读来的数据，这个数据不能存放在通用寄存器中，否则会破坏其原有内容。

如：两个操作数分别来自主存和$R_0$，最后结果存回$R_0$，那么从主存中取来的操作数直接放入暂存器，就不会破坏运算前$R_0$的内容。

结构简单，容易实现，但数据传输存在较多冲突的现象，性能较低。



## 运算器的基本结构

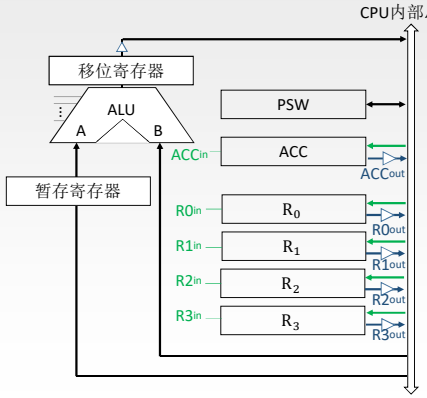



1. 算术逻辑单元：主要功能是进行算术/逻辑运算。
2. 通用寄存器组：如AX、BX、CX、DX、SP等，用于存放操作数（包括源操作数、目的操作数及中间结果）和各种地址信息等。SP是堆栈指针，用于指示栈顶的地址。
3. 暂存寄存器：用于暂存从主存读来的数据，这个数据不能存放在通用寄存器中，否则会破坏其原有内容。

如：两个操作数分别来自主存和$R_0$，最后结果存回$R_0$，那么从主存中取来的操作数直接放入暂存器，就不会破坏运算前$R_0$的内容。

结构简单，容易实现，但数据传输存在较多冲突的现象，性能较低。

## 运算器的基本结构



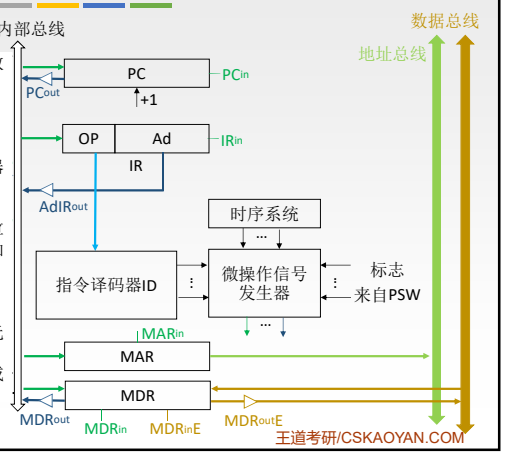

1. 算术逻辑单元：主要功能是进行算术/逻辑运算。
2. 通用寄存器组：如AX、BX、CX、DX、SP等，用于存放操作数（包括源操作数、目的操作数及中间结果）和各种地址信息等。SP是堆栈指针，用于指示栈顶的地址。
3. 暂存寄存器：用于暂存从主存读来的数据，这个数据不能存放在通用寄存器中，否则会破坏其原有内容。
4. 累加寄存器：它是一个通用寄存器，用于暂时存放ALU运算的结果信息，用于实现加法运算。
5. 程序状态字寄存器：保留由算术逻辑运算指令或测试指令的结果而建立的各种状态信息，如溢出标志（OP）、符号标志（SF）、零标志（ZF）、进位标志（CF）等。PSW中的这些位参与并决定微操作的形成。
6. 移位器：对运算结果进行移位运算。
7. 计数器：控制乘除运算的操作步数。

结构简单，容易实现，但数据传输存在较多冲突的现象，性能较低。



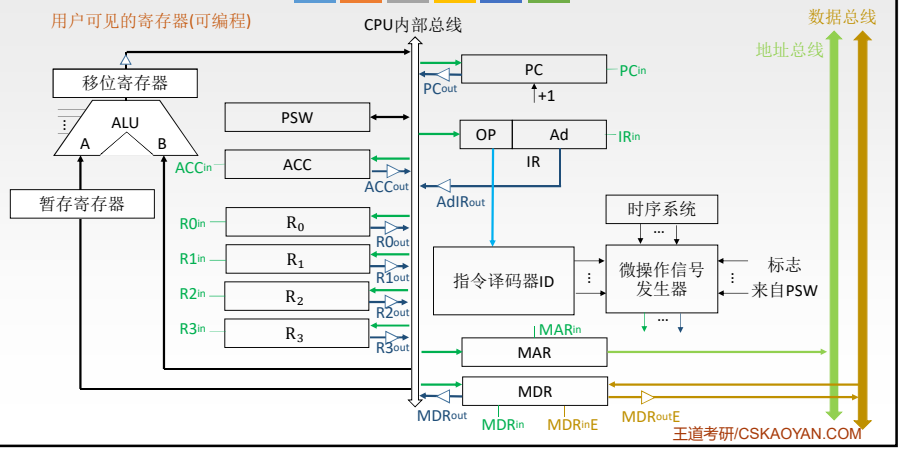
## 控制器的基本结构

1. 程序计数器：用于指出下一条指令在主存中的存放地址。CPU就是根据PC的内容去主存中取指令的。因程序中指令（通常）是顺序执行的，所以PC有自增功能。
2. 指令寄存器：用于保存当前正在执行的那条指令。
3. 指令译码器：仅对操作码字段进行译码，向控制器提供特定的操作信号。
4. 微操作信号发生器：根据IR的内容（指令）、PSW的内容（状态信息）及时序信号，产生控制整个计算机系统所需的各种控制信号，其结构有组合逻辑型和存储逻辑型两种。
5. 时序系统：用于产生各种时序信号，它们都是由统一时钟（CLOCK）分频得到。
6. 存储器地址寄存器：用于存放所要访问的主存单元的地址。
7. 存储器数据寄存器：用于存放向主存写入的信息或从主存中读出的信息。



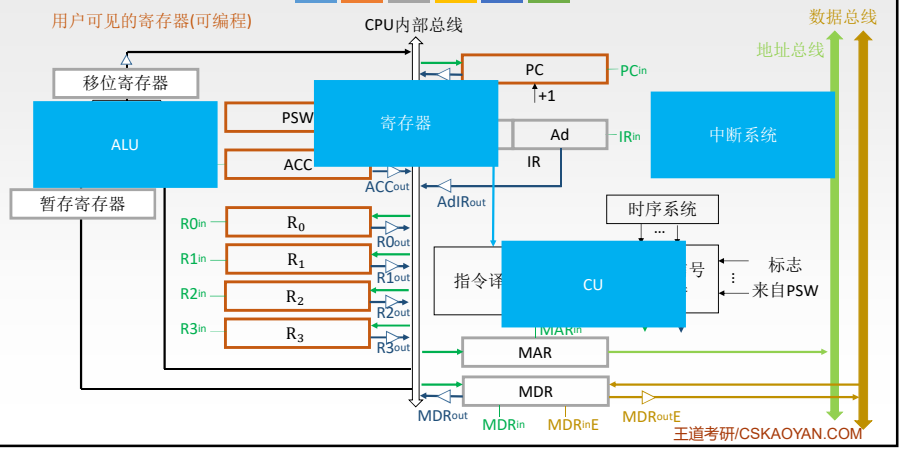



## CPU的基本结构

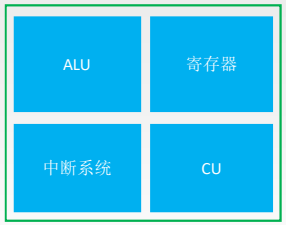





## CPU的基本结构

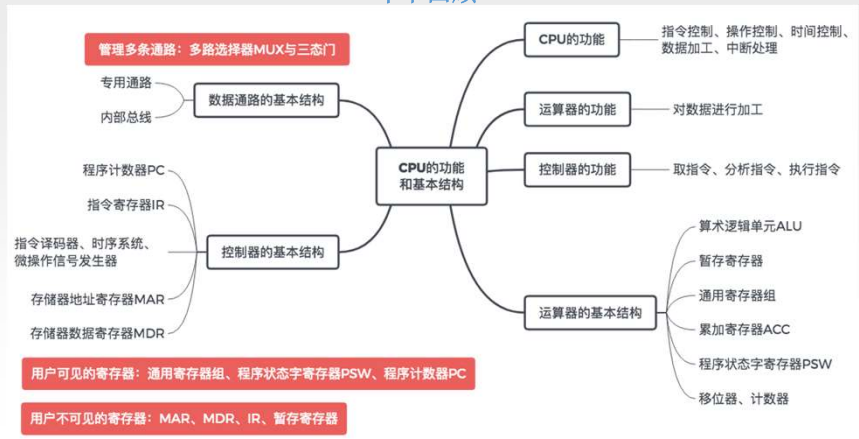


## CPU的基本结构






## 本节回顾





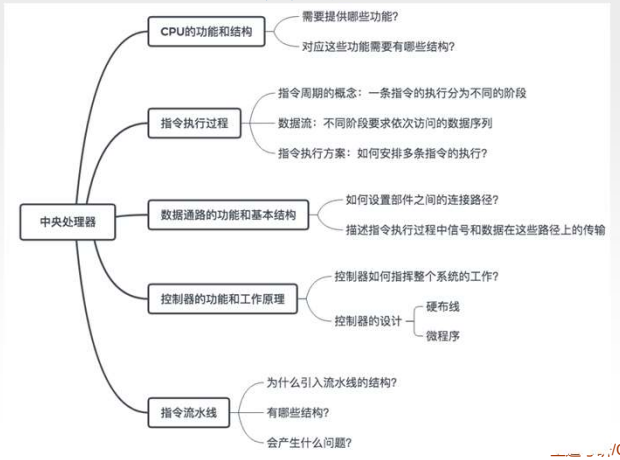

本节内容

# 中央处理器

# 指令执行过程



## 本章总览

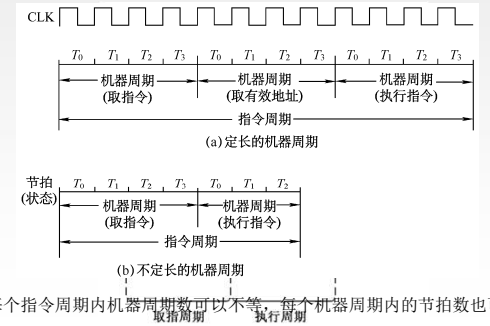


## 指令周期

**指令周期**：CPU从主存中每取出并执行一条指令所需的全部时间。

**指令周期**常常用若干**机器周期**来表示，机器周期又叫**CPU周期**。

一个**机器周期**又包含若干**时钟周期**（也称为**节拍**、**T周期**或**CPU时钟周期**，它是CPU操作的**最基本单位**）。

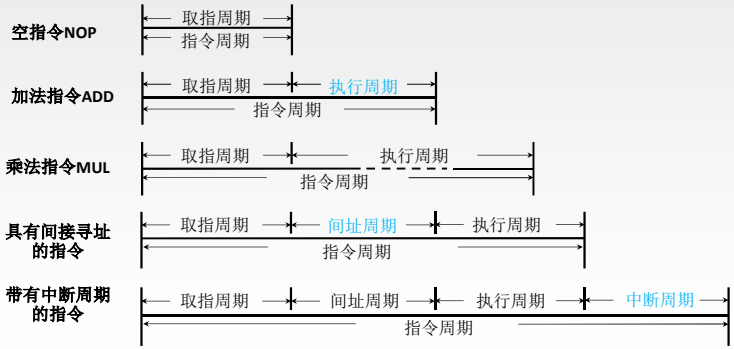



每个指令周期内机器周期数可以不等，每个机器周期内的节拍数也可以不等。



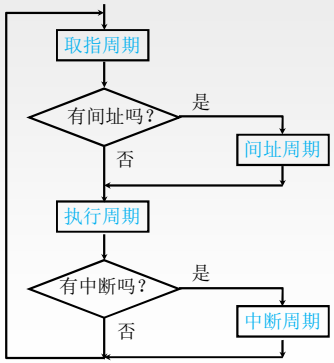
## 指令周期

每个指令周期内机器周期数可以不等，每个机器周期内的节拍数也可以不等。



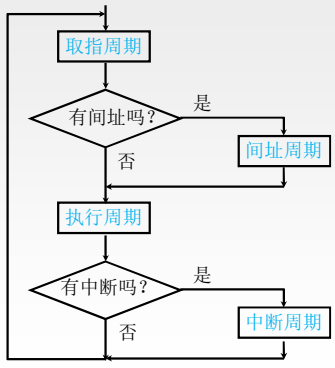


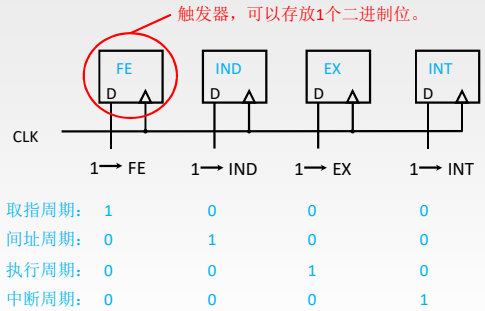

## 指令周期流程






## 指令周期流程

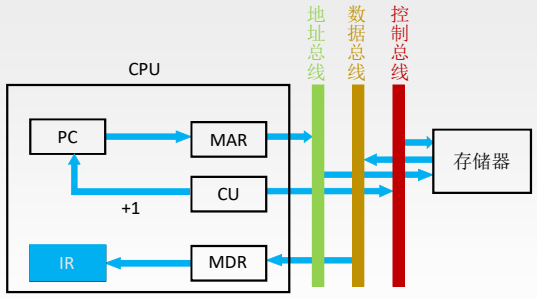



| | | | | |
|---|---|---|---|---|
| 取指周期： | 1 | 0 | 0 | 0 |
| 间址周期： | 0 | 1 | 0 | 0 |
| 执行周期： | 0 | 0 | 1 | 0 |
| 中断周期： | 0 | 0 | 0 | 1 |

四个工作周期都有CPU访存操作，只是访存的目的不同。**取指周期**是为了**取指令**，**间址周期**是为了**取有效地址**，**执行周期**是为了**取操作数**，**中断周期**是为了**保存程序断点**。

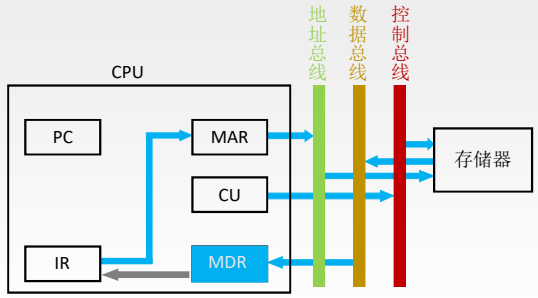
## 指令周期的数据流-取指周期




1. 当前指令地址送至存储器地址寄存器，记做：(PC) → MAR
2. CU发出控制信号，经控制总线传到主存，这里是**读信号**，记做：1 → R
3. 将MAR所指主存中的内容经数据总线送入MDR，记做：M(MAR) → MDR
4. 将MDR中的内容(此时是指令)送入IR，记做：(MDR) → IR
5. CU发出控制信号，形成下一条指令地址，记做：(PC)+1 → PC



## 指令周期的数据流-间址周期




1. 将指令的地址码送入MAR，
记做：Ad(IR) → MAR
或Ad(MDR) → MAR
2. CU发出控制信号，启动主存做**读操作**，记做：1 → R
3. 将MAR所指主存中的内容经数据总线送入MDR，记做：M(MAR) → MDR
4. 将有效地址送至指令的地址码字段，记做：MDR→ Ad(IR)

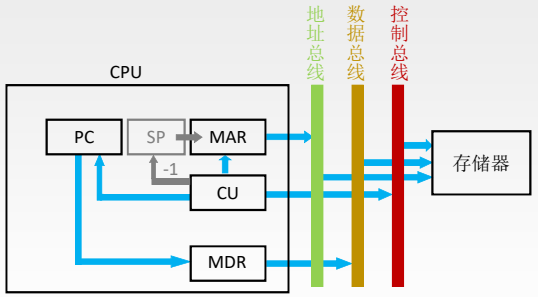


## 指令周期的数据流-执行周期

执行周期的任务是根据IR中的指令字的操作码和操作数通过ALU操作产生执行结果。
不同指令的执行周期操作不同，因此没有统一的数据流向。



## 指令周期的数据流-中断周期

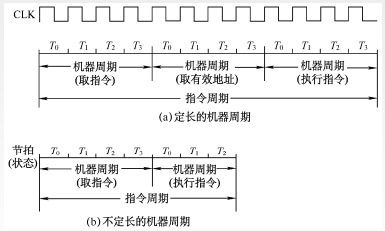



中断：暂停当前任务去完成其他任务。
为了能够恢复当前任务，需要**保存断点**。
一般使用堆栈来保存断点，这里用SP表示栈顶地址，假设**SP指向栈顶元素**，进栈操作是**先修改指针，后存入数据**。

1. CU控制将SP减1，修改后的地址送入MAR
记做：(SP)-1 → SP，(SP) → MAR
本质上是将断点存入某个存储单元，假设其地址为a，故可记做：a → MAR
2. CU发出控制信号，启动主存做**写操作**，记做：1 → W
3. 将断点(PC内容) 送入MDR，
记做：(PC) → MDR
4. CU控制将中断服务程序的入口地址(由向量地址形成部件产生)送入PC，
记做：向量地址→ PC

## 指令执行方案

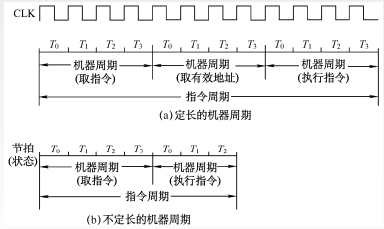
一个指令周期通常要包括几个时间段（执行步骤），每个步骤完成指令的一部分功能，几个依次执行的步骤完成这条指令的全部功能。






## 指令执行方案

一个指令周期通常要包括几个时间段（执行步骤），每个步骤完成指令的一部分功能，几个依次执行的步骤完成这条指令的全部功能。



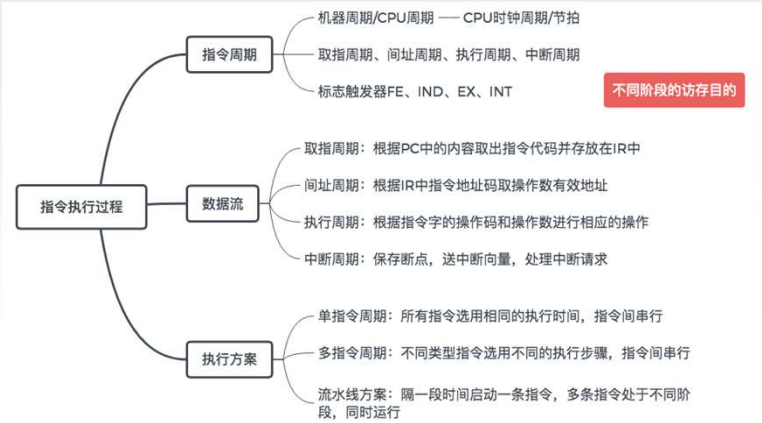

**方案1．单指令周期**
对所有指令都选用相同的执行时间来完成。
指令之间串行执行；指令周期取决于执行时间最长的指令的执行时间。
对于那些本来可以在更短时间内完成的指令，要使用这个较长的周期来完成，会降低整个系统的运行速度。

**方案2．多指令周期**
对不同类型的指令选用不同的执行步骤来完成。
指令之间串行执行；可选用不同个数的时钟周期来完成不同指令的执行过程。
需要更复杂的硬件设计。

**方案3．流水线方案**
在每一个时钟周期启动一条指令，尽量让多条指令同时运行，但各自处在不同的执行步骤中。
指令之间并行执行。



## 本节回顾






本节内容

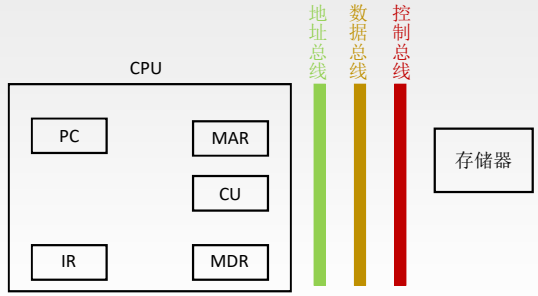
# 中央处理器

# 数据通路的功能和基本结构
# 单总线

## 本章总览

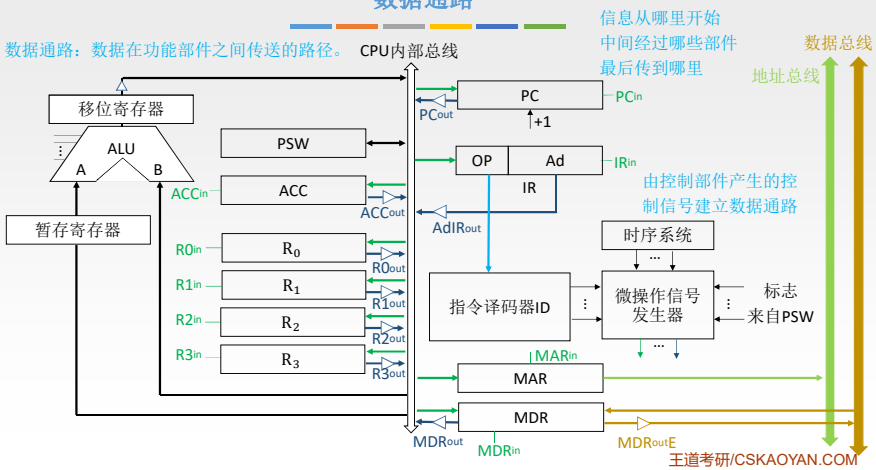



## 指令周期的数据流



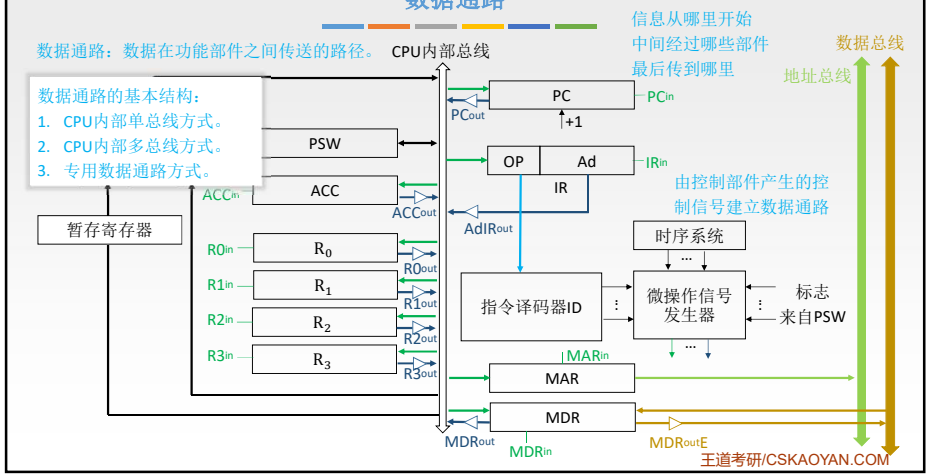



## 数据通路

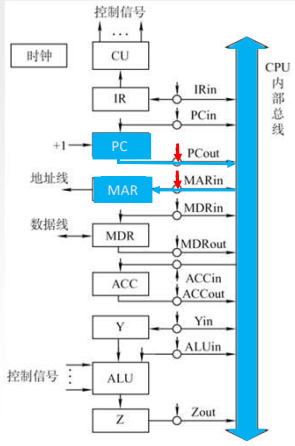





## 数据通路



## 数据通路-CPU内部单总线方式



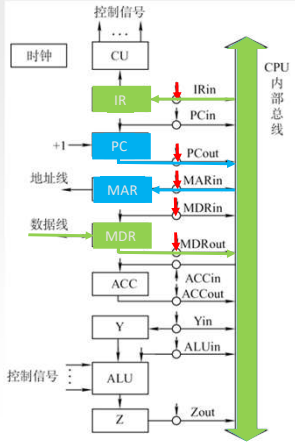

**内部总线**是指同一部件，如CPU内部连接各寄存器及运算部件之间的总线；
**系统总线**是指同一台计算机系统的各部件，如CPU、内存、通道和各类I/O接口间互相连接的总线。

**1. 寄存器之间数据传送**
比如把PC内容送至MAR，实现传送操作的流程及控制信号为：
(PC)→Bus　　PCout有效，PC内容送总线
Bus→MAR　　MARin有效，总线内容送MAR

**关于是否加括号见王道p7最下方的讨论，写数据通路时，更关心数据流经的路径，可以省略括号，但从真题来看，考微操作序列居多，所以建议都加上，题目给出示例时跟题目保持一致。**



## 数据通路-CPU内部单总线方式



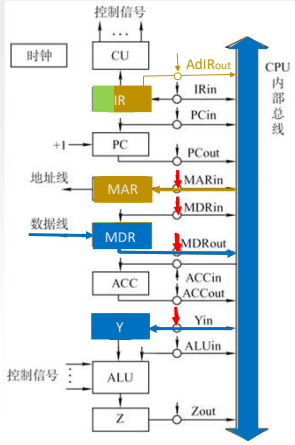

**内部总线**是指同一部件，如CPU内部连接各寄存器及运算部件之间的总线；
**系统总线**是指同一台计算机系统的各部件，如CPU、内存、通道和各类I/O接口间互相连接的总线。

**1. 寄存器之间数据传送**
比如把PC内容送至MAR，实现传送操作的流程及控制信号为：
(PC)→Bus　　PCout有效，PC内容送总线
Bus→MAR　　MARin有效，总线内容送MAR

**2. 主存与CPU之间的数据传送**
比如CPU从主存读取指令，实现传送操作的流程及控制信号为：
(PC)→Bus→MAR　　PCout和MARin有效，现行指令地址→MAR
1→R　　CU发读命令(通过控制总线发出，图中未画出)
MEM(MAR)→MDR　　MDRin有效
MDR→Bus→IR　　MDRout和IRin有效，现行指令→IR



## 数据通路-CPU内部单总线方式



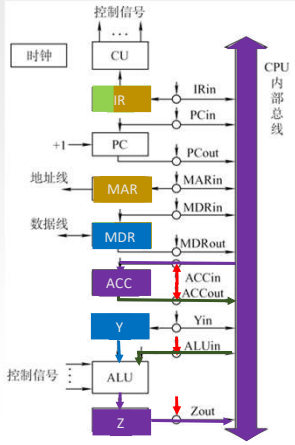

**1. 寄存器之间数据传送**
比如把PC内容送至MAR，实现传送操作的流程及控制信号为：
(PC)→Bus　　PCout有效，PC内容送总线
Bus→MAR　　MARin有效，总线内容送MAR

**2. 主存与CPU之间的数据传送**
比如CPU从主存读取指令，实现传送操作的流程及控制信号为：
(PC)→Bus→MAR　　PCout和MARin有效，现行指令地址→MAR
1→R　　CU发读命令(通过控制总线发出，图中未画出)
MEM(MAR)→MDR　　MDRin有效
MDR→Bus→IR　　MDRout和IRin有效，现行指令→IR

**3. 执行算术或逻辑运算**
比如一条加法指令，微操作序列及控制信号为：
Ad(IR)→Bus→MAR　　MDRout和MARin有效　或AdIRout和MARin有效
1→R　　CU发读命令
MEM(MAR)→数据线→MDR　MDRin有效
MDR→Bus→Y　　MDRout和Yin有效，操作数→Y



## 数据通路-CPU内部单总线方式

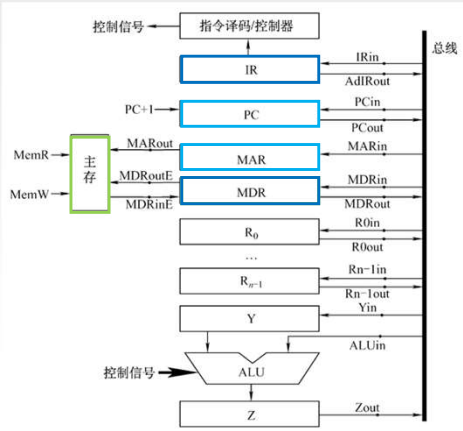



**1. 寄存器之间数据传送**
比如把PC内容送至MAR，实现传送操作的流程及控制信号为：
(PC)→Bus　　PCout有效，PC内容送总线
Bus→MAR　　MARin有效，总线内容送MAR

**2. 主存与CPU之间的数据传送**
比如CPU从主存读取指令，实现传送操作的流程及控制信号为：
(PC)→Bus→MAR　　PCout和MARin有效，现行指令地址→MAR
1→R　　CU发读命令(通过控制总线发出，图中未画出)
MEM(MAR)→MDR　　MDRin有效
MDR→Bus→IR　　MDRout和IRin有效，现行指令→IR

**3. 执行算术或逻辑运算**
比如一条加法指令，微操作序列及控制信号为：
Ad(IR)→Bus→MAR　　MDRout和MARin有效
1→R　　CU发读命令
MEM(MAR)→数据线→MDR　MDRin有效
MDR→Bus→Y　　MDRout和Yin有效，操作数→Y
(ACC)+(Y)→Z　　ACCout和ALUin有效，CU向ALU发送加命令
Z→ACC　　Zout和ACCin有效，结果→ACC

## CPU内部单总线方式-例题




设有如图所示的单总线结构，分析指令ADD (R0)，R1 的指令流程和控制信号。

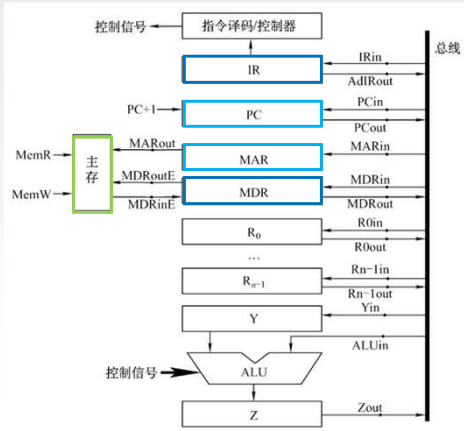
1. 分析指令功能和指令周期
功能：((R0))+(R1)→(R0)
取指周期、间址周期、执行周期

2. 写出各阶段的指令流程
取指周期：公共操作

| 时序 | 微操作 | 有效控制信号 |
|---|---|---|
| 1 | (PC)→MAR | PCout，MARin |
| 2 | M(MAR)→MDR | MemR，MARout，MDRinE |
| 3 | (MDR)→IR | MDRout，IRin |
| 4 | 指令译码 | - |
| 5 | (PC)+1→PC | - |



## CPU内部单总线方式-例题




设有如图所示的单总线结构，分析指令ADD (R0)，R1 的指令流程和控制信号。

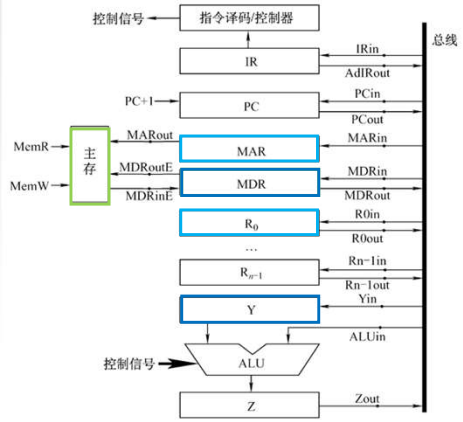
1. 分析指令功能和指令周期
功能：((R0))+(R1)→(R0)
取指周期、间址周期、执行周期

2. 写出各阶段的指令流程
取指周期：公共操作

| 时序 | 微操作 | 有效控制信号 |
|---|---|---|
| 1 | (PC)→MAR | PCout，MARin |
| 2 | M(MAR)→MDR<br>(PC)+1→PC | MemR，MARout，MDRinE |
| 3 | (MDR)→IR | MDRout，IRin |
| 4 | 指令译码 | - |



## CPU内部单总线方式-例题




设有如图所示的单总线结构，分析指令ADD (R0)，R1 的指令流程和控制信号。

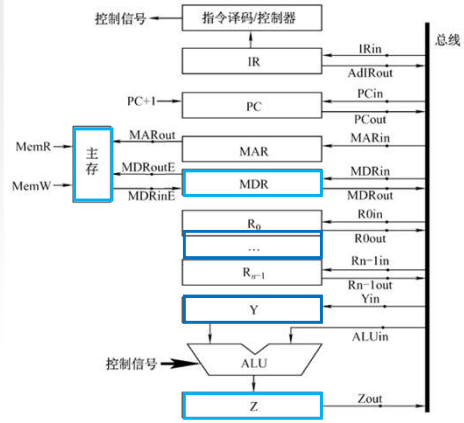
1. 分析指令功能和指令周期
功能：((R0))+(R1)→(R0)
取指周期、间址周期、执行周期

2. 写出各阶段的指令流程
间址周期：完成取数操作，被加数在主存中，加数已经放在寄存器R1中。

| 时序 | 微操作 | 有效控制信号 |
|---|---|---|
| 1 | (R0)→MAR | R0out，MARin |
| 2 | M(MAR)→MDR | MemR，MARout，MDRinE |
| 3 | (MDR)→Y | MDRout，Yin |



## CPU内部单总线方式-例题




设有如图所示的单总线结构，分析指令ADD (R0)，R1 的指令流程和控制信号。

1. 分析指令功能和指令周期
功能：((R0))+(R1)→(R0)
取指周期、间址周期、执行周期

2. 写出各阶段的指令流程
执行周期：完成取数操作，被加数在主存中，加数已经放在寄存器R1中。

| 时序 | 微操作 | 有效控制信号 |
|---|---|---|
| 1 | (R1)+(Y)→Z | R1out，ALUin，CU向ALU发ADD控制信号 |
| 2 | (Z)→MDR | Zout，MDRin |
| 3 | (MDR)→M(MAR) | MemW，MDRoutE，MARout |

## 本节回顾

- 内部总线与系统总线
- 各阶段的微操作序列和控制信号
- 数据通路
  - CPU内部总线
    - 单总线
    - 多总线
    - ALU需要配合暂存器使用
  - 专用数据通路



本节内容

# 中央处理器

## 数据通路的功能和基本结构
## 专用通路

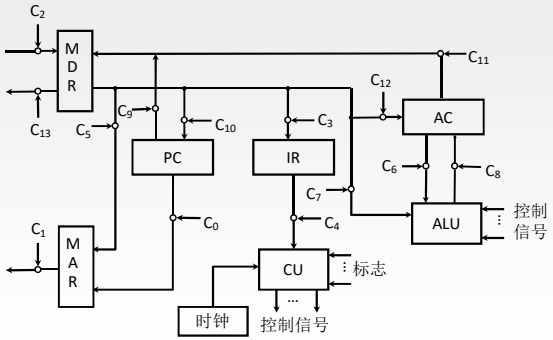


## 上节回顾

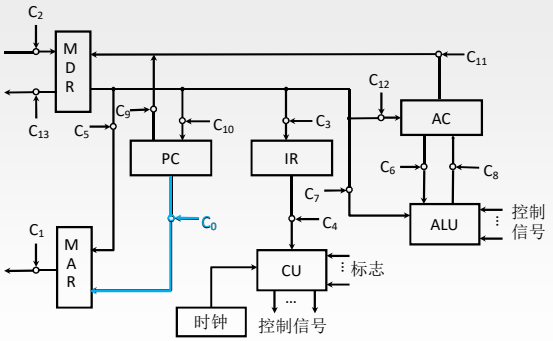
- 内部总线与系统总线
- 各阶段的微操作序列和控制信号
- 数据通路
  - CPU内部总线
    - 单总线
    - 多总线
    - ALU需要配合暂存器使用
  - 专用数据通路



## 专用数据通路方式



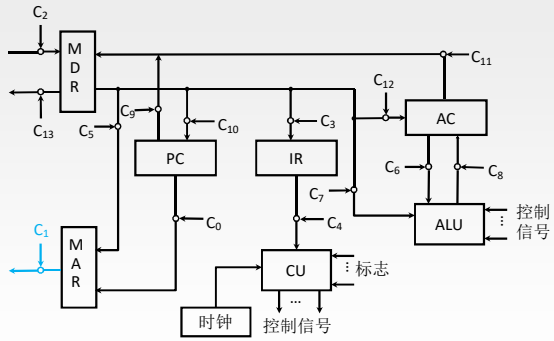
## 专用数据通路方式-取指周期




(PC)→MAR　　C0有效



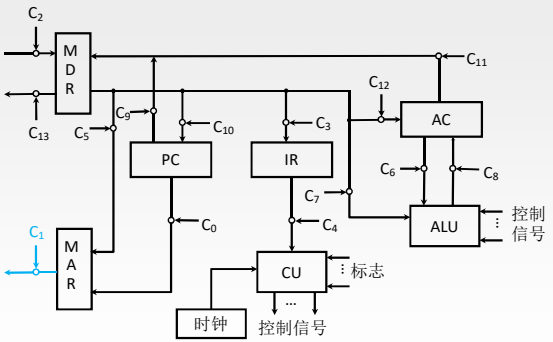
## 专用数据通路方式-取指周期




(PC)→MAR　　C0有效
(MAR)→主存　　C1有效



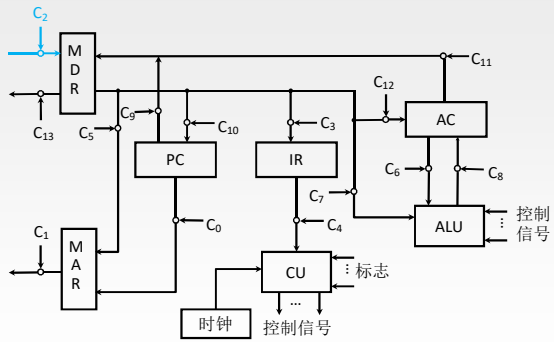
## 专用数据通路方式-取指周期




(PC)→MAR　　C0有效
(MAR)→主存　　C1有效
1→R　　控制单元向主存发送读命令

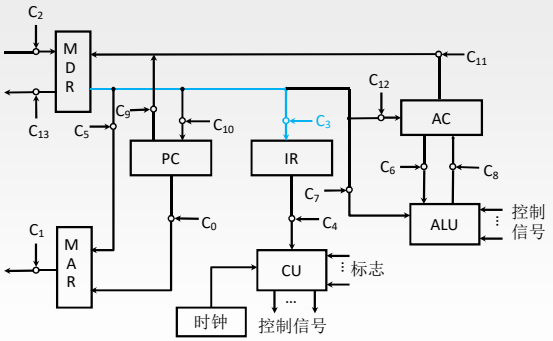


## 专用数据通路方式-取指周期




(PC)→MAR　　C0有效
(MAR)→主存　　C1有效
1→R　　控制单元向主存发送读命令
M(MAR)→MDR　　C2有效

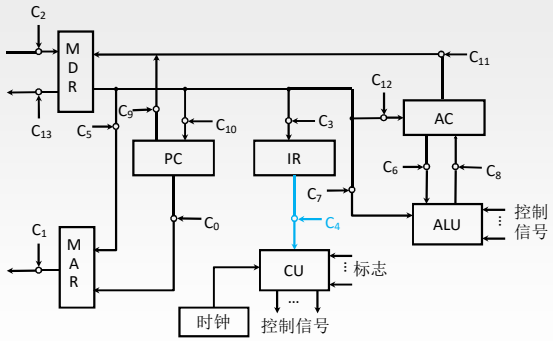
## 专用数据通路方式-取指周期




(PC)→MAR　　C0有效
(MAR)→主存　　C1有效
1→R　　控制单元向主存发送读命令
M(MAR)→MDR　　C2有效
(MDR)→IR　　C3有效
(PC)+1→PC



## 专用数据通路方式-取指周期



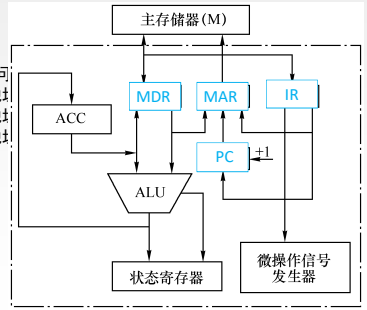

(PC)→MAR　　C0有效
(MAR)→主存　　C1有效
1→R　　控制单元向主存发送读命令
M(MAR)→MDR　　C2有效
(MDR)→IR　　C3有效
(PC)+1→PC
Op(IR)→CU　　C4有效



## 专用数据通路方式-例题

下图是一个简化了的CPU与主存连接结构示意图（图中省略了所有的多路选择器）。其中有一个累加寄存器（ACC）、一个状态数据寄存器和其他4个寄存器：主存地址寄存器（MAR）、主存数据寄存器（MDR）、程序寄存器（PC）和指令寄存器（IR），各部件及其之间的连线表示数据通路，箭头表示信息传递方向。

要求：
（1）请写出图中a、b、c、d 4个寄存器的名称。
（2）简述图中取指令的数据通路。
（3）简述数据在运算器和主存之间进行存/取访问
（4）简述完成指令LDA X的数据通路（X为主存地址
（5）简述完成指令ADD Y的数据通路（Y为主存地址
（6）简述完成指令STA Z的数据通路（Z为主存地址



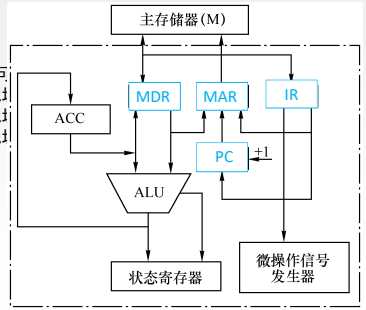

(1)
d能自动“+1”，是PC
PC内容是地址，送MAR，故c是MAR
b与微操作信号发生器相连，是IR
与主存相连的寄存器是MAR和MDR，c是MAR，则a是MDR



## 专用数据通路方式-例题

下图是一个简化了的CPU与主存连接结构示意图（图中省略了所有的多路选择器）。其中有一个累加寄存器（ACC）、一个状态数据寄存器和其他4个寄存器：主存地址寄存器（MAR）、主存数据寄存器（MDR）、程序寄存器（PC）和指令寄存器（IR），各部件及其之间的连线表示数据通路，箭头表示信息传递方向。

要求：
（1）请写出图中a、b、c、d 4个寄存器的名称。
（2）简述图中取指令的数据通路。
（3）简述数据在运算器和主存之间进行存/取访问
（4）简述完成指令LDA X的数据通路（X为主存地址
（5）简述完成指令ADD Y的数据通路（Y为主存地址
（6）简述完成指令STA Z的数据通路（Z为主存地址




(2)
(PC) → MAR
M(MAR) → MDR
(MDR) → IR

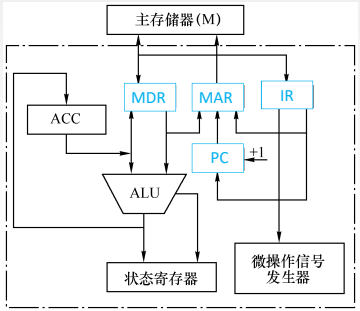
## 专用数据通路方式-例题

下图是一个简化了的CPU与主存连接结构示意图（图中省略了所有的多路选择器）。其中有一个累加寄存器（ACC）、一个状态数据寄存器和其他4个寄存器：主存地址寄存器（MAR）、主存数据寄存器（MDR）、程序寄存器（PC）和指令寄存器（IR），各部件及其之间的连线表示数据通路，箭头表示信息传递方向。

（3）简述数据在运算器和主存之间进行存/取访问的数据通路。

存/取的数据放到ACC中
设数据地址已放入MAR
取：
M(MAR) → MDR
(MDR) → ALU → ACC

存：
(ACC) → MDR
(MDR) → M(MAR)



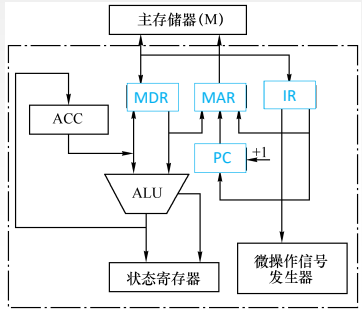



## 专用数据通路方式-例题

下图是一个简化了的CPU与主存连接结构示意图（图中省略了所有的多路选择器）。其中有一个累加寄存器（ACC）、一个状态数据寄存器和其他4个寄存器：主存地址寄存器（MAR）、主存数据寄存器（MDR）、程序寄存器（PC）和指令寄存器（IR），各部件及其之间的连线表示数据通路，箭头表示信息传递方向。

（4）简述完成指令LDA X的数据通路（X为主存地址，LDA的功能为(X)→ACC）。

X → MAR
M(MAR) → MDR
(MDR) → ALU → ACC



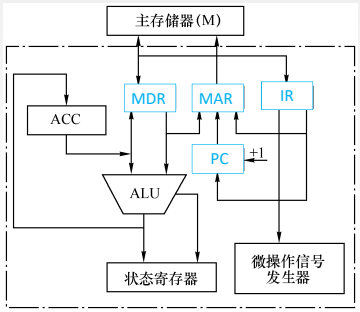



## 专用数据通路方式-例题

下图是一个简化了的CPU与主存连接结构示意图（图中省略了所有的多路选择器）。其中有一个累加寄存器（ACC）、一个状态数据寄存器和其他4个寄存器：主存地址寄存器（MAR）、主存数据寄存器（MDR）、程序寄存器（PC）和指令寄存器（IR），各部件及其之间的连线表示数据通路，箭头表示信息传递方向。

（5）简述完成指令ADD Y的数据通路（Y为主存地址，ADD的功能为(ACC)+(Y)→ACC）。

Y → MAR
M(MAR) → MDR
(MDR) → ALU, (ACC) → ALU
ALU → ACC



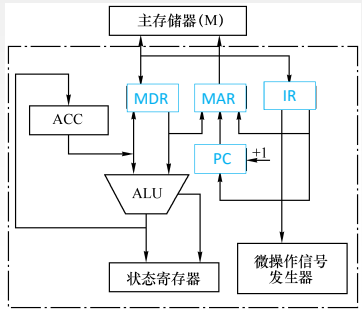



## 专用数据通路方式-例题

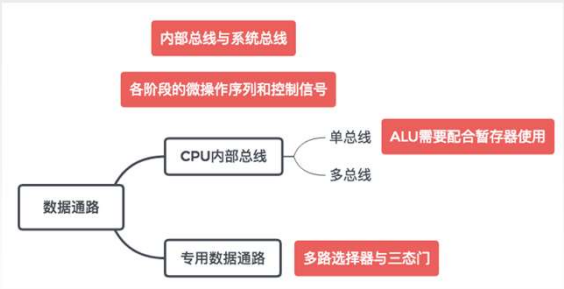
下图是一个简化了的CPU与主存连接结构示意图（图中省略了所有的多路选择器）。其中有一个累加寄存器（ACC）、一个状态数据寄存器和其他4个寄存器：主存地址寄存器（MAR）、主存数据寄存器（MDR）、程序寄存器（PC）和指令寄存器（IR），各部件及其之间的连线表示数据通路，箭头表示信息传递方向。

（6）简述完成指令STA Z的数据通路（Z为主存地址，STA的功能为(ACC)→Z）。

Z → MAR
(ACC) → MDR
(MDR) → M(MAR)



## 本节回顾




涉及的主要操作类型：
寄存器之间的数据传送；
主存与CPU之间的数据传送；
使用ALU进行算术逻辑运算。

基本思路：
利用题目提供的数据通路进行数据传送；
由CU发出的控制信号实现通路的建立。



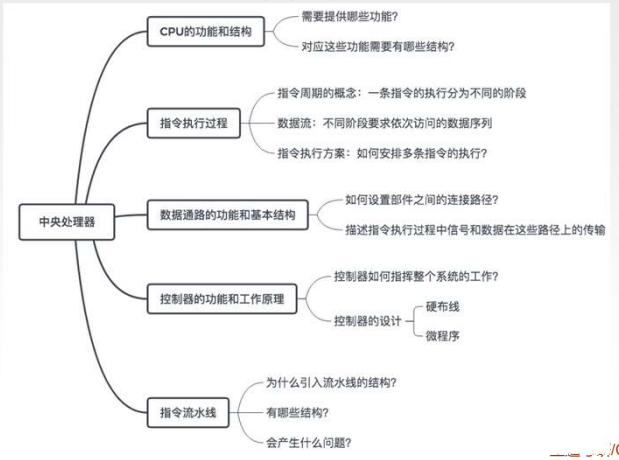
本节内容

# 中央处理器

## 控制器的功能和工作原理
## 硬布线

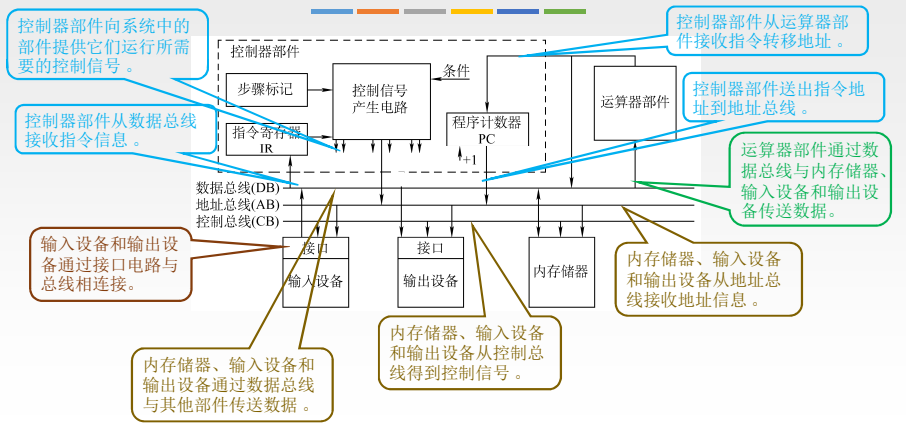


## 本章总览






## 控制器的结构和功能



## 控制器的结构和功能

控制器是计算机系统的指挥中心，控制器的主要功能有：
1）从主存中取出一条指令，并指出下一条指令在主存中的位置。
2）对指令进行译码或测试，产生相应的操作控制信号，以便启动规定的动作。
3）指挥并控制CPU、主存、输入和输出设备之间的数据流动方向。



## 控制单元的输入和输出

1. 输入

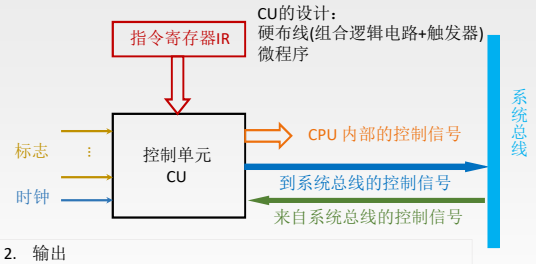
(1)指令寄存器 OP(IR) —> CU
控制信号的产生与操作码有关

(2)时钟
一个时钟脉冲发一个操作命令或一组需要同时执行的操作命令

(3)标志
如条件转移指令，根据相应的标志位决定下一步操作

(4)外来信号
如：中断请求信号INTR
总线请求信号HRQ

CU的设计：
硬布线(组合逻辑电路+触发器)
微程序

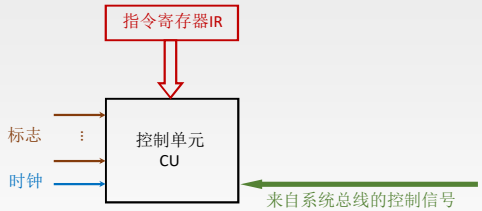



2. 输出
(1) CPU 内部的控制信号
寄存器之间的数据传输、PC的修改、控制ALU进行相应的运算

(2)到控制总线的控制信号
到存储器：访存控制信号$\overline{MREQ}$、读命令$\overline{RD}$ 、写命令$\overline{WR}$
到I/O设备：访问I/O设备的控制信号$\overline{IO}$
中断响应信号INTA、总线响应信号HLDA

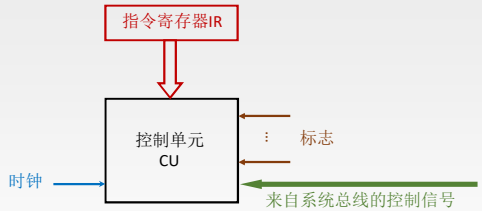


## 硬布线控制器

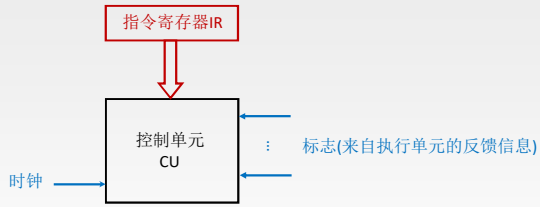





## 硬布线控制器

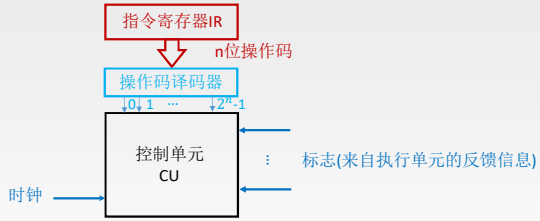


## 硬布线控制器



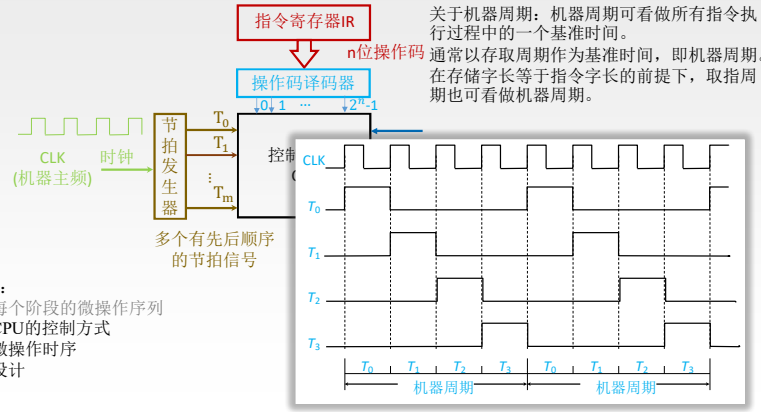



## 硬布线控制器






## 硬布线控制器




关于机器周期：机器周期可看做所有指令执行过程中的一个基准时间。
通常以存取周期作为基准时间，即机器周期。
在存储字长等于指令字长的前提下，取指周期也可看做机器周期。

设计步骤：
1. 分析每个阶段的微操作序列
2. 选择CPU的控制方式
3. 安排微操作时序
4. 电路设计

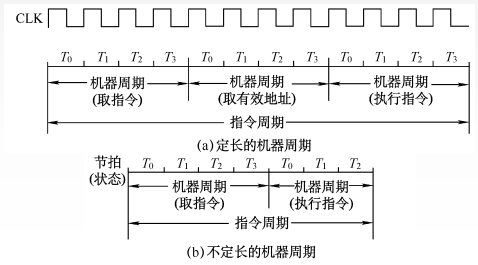


## CPU的控制方式

CPU的控制方式：产生不同微操作命令序列所用的时序控制方式。

**1. 同步控制方式**

整个系统所有的控制信号均来自一个**统一的时钟信号**。
通常以最长的微操作序列和最烦琐的微操作作为标准，采取完全统一的、具有相同时间间隔和相同数目的节拍作为机器周期来运行不同的指令。
同步控制方式的优点是控制电路简单，缺点是运行速度慢。



## CPU的控制方式

CPU的控制方式：产生不同微操作命令序列所用的时序控制方式。

**1. 同步控制方式**
整个系统所有的控制信号均来自一个**统一的时钟信号**。
同步控制方式的优点是控制电路简单，缺点是运行速度慢。

**2. 异步控制方式**
异步控制方式**不存在基准时标信号**。
各部件按自身固有的速度工作，通过**应答方式**进行联络。
异步控制方式的优点是运行速度快，缺点是控制电路比较复杂。

**3. 联合控制方式**
对各种不同的指令的微操作实行**大部分采用同步控制、小部分采用异步控制**的办法。

设计步骤：
1. 分析每个阶段的微操作序列
2. 选择CPU的控制方式　　假设采用同步控制方式，
3. 安排微操作时序　　一个机器周期内安排3个节拍(时钟周期)。
4. 电路设计



## 安排微操作时序的原则

原则一　微操作的 先后顺序不得 随意 更改
原则二　被控对象不同 的微操作
　　尽量安排在 一个节拍 内完成
原则三　占用 时间较短 的微操作
　　尽量 安排在 一个节拍 内完成
　　并允许有先后顺序



## 安排微操作时序-取指周期

原则一　微操作的 先后顺序不得 随意 更改
原则二　被控对象不同 的微操作
　　尽量安排在 一个节拍 内完成
原则三　占用 时间较短 的微操作
　　尽量 安排在 一个节拍 内完成
　　并允许有先后顺序

(1) PC → MAR
(2) 1 → R　　存储器空闲即可
(3) M ( MAR ) → MDR　　在(1)之后
(4) MDR → IR　　在(3)之后
(5) OP ( IR ) → ID　　在(4)之后
(6) ( PC ) + 1 → PC　　在(1)之后



## 安排微操作时序-取指周期

原则一　微操作的 先后顺序不得 随意 更改
原则二　被控对象不同 的微操作
　　尽量安排在 一个节拍 内完成
原则三　占用 时间较短 的微操作
　　尽量 安排在 一个节拍 内完成
　　并允许有先后顺序

$T_0$　(1) PC → MAR
$T_0$　(2) 1 → R　　存储器空闲即可
$T_1$　(3) M ( MAR ) → MDR　　在(1)之后
$T_1$　(6) ( PC ) + 1 → PC　　在(1)之后
$T_2$　(4) MDR → IR　　在(3)之后
$T_2$　(5) OP ( IR ) → ID　　在(4)之后

两个微操作占用时间较短，根据原则三安排在一个节拍

## 安排微操作时序-间址周期

原则一 微操作的 先后顺序不得 随意 更改

原则二 被控对象不同 的微操作

尽量安排在 一个节拍 内完成

原则三 占用 时间较短 的微操作

尽量 安排在 一个节拍 内完成

并允许有先后顺序

$T_0$ (1) Ad(IR) → MAR

$T_0$ (2) 1 → R

$T_1$ (3) M ( MAR ) → MDR

$T_2$ (4) MDR → Ad(IR)



## 安排微操作时序-执行周期

原则一 微操作的 先后顺序不得 随意 更改

原则二 被控对象不同 的微操作

尽量安排在 一个节拍 内完成

原则三 占用 时间较短 的微操作

尽量 安排在 一个节拍 内完成

并允许有先后顺序

① CLA

clear ACC清零

$T_0$ $T_1$ $T_2$ 0 → AC

② COM

complement ACC取反

$T_0$ $T_1$ $T_2$ $\overline{AC}$ → AC

③ SHR

shift 算术右移

$T_0$ $T_1$ $T_2$ L ( AC ) → R ( AC )

$T_2$ $AC_0$ → $AC_0$

④ CSL

cyclic shift 循环左移

$T_0$ $T_1$ $T_2$ R ( AC ) → L ( AC )， $AC_0$ → $AC_n$

⑤ STP

stop 停机

$T_0$ $T_1$ $T_2$ 0 → G



## 安排微操作时序-执行周期

(1) 非访存指令

① CLA

clear ACC清零

$T_0$ $T_1$ $T_2$ 0 → AC

② COM

complement ACC取反

$T_0$ $T_1$ $T_2$ $\overline{AC}$ → AC

③ SHR

shift 算术右移

$T_0$ $T_1$ $T_2$ L ( AC ) → R ( AC )

$T_2$ $AC_0$ → $AC_0$

④ CSL

cyclic shift 循环左移

$T_0$ $T_1$ $T_2$ R ( AC ) → L ( AC )， $AC_0$ → $AC_n$

⑤ STP

stop 停机

$T_0$ $T_1$ $T_2$ 0 → G

(2) 访存指令

⑥ ADD X 加法指令 隐含ACC

$T_0$ Ad ( IR ) → MAR， 1 → R

$T_1$ M ( MAR ) → MDR

$T_2$ ( AC ) + ( MDR ) → AC

⑦ STA X 存数指令 隐含ACC

$T_0$ Ad ( IR ) → MAR， 1 → W

$T_1$ AC → MDR

$T_2$ MDR → M ( MAR )

⑧ LDA X 取数指令 隐含ACC

$T_0$ Ad ( IR ) → MAR， 1 → R

$T_1$ M ( MAR ) → MDR

$T_2$ MDR → AC

(3) 转移指令

⑨ JMP X jump 无条件转移

$T_0$ $T_1$ $T_2$ Ad ( IR ) → PC

⑩ BAN X Branch ACC Negative 条件转移

$T_0$ $T_1$ $T_2$ $A_0 \bullet$ Ad ( IR ) + $\overline{A_0} \bullet$ PC → PC



## 安排微操作时序-中断周期

原则一 微操作的 先后顺序不得 随意 更改

原则二 被控对象不同 的微操作

尽量安排在 一个节拍 内完成

原则三 占用 时间较短 的微操作

尽量 安排在 一个节拍 内完成

并允许有先后顺序

设计步骤：

1. 分析每个阶段的微操作序列
2. 选择CPU的控制方式
3. 安排微操作时序
4. 电路设计

$T_0$ (1) a → MAR

$T_0$ (2) 1 → W 存储器空闲即可

$T_0$ (3) 0 → EINT 硬件关中断

$T_1$ (4) ( PC ) → MDR 内部数据通路空闲即可

$T_2$ (5) MDR → M(MAR) 在(3)之后

$T_2$ (6) 向量地址 → PC 在(3)之后

这些操作由中断隐指令完成

注：中断隐指令不是一条指令，而是指一条指令的中断周期由硬件完成的一系列操作

中断周期的三个任务：

1. 保存断点
2. 形成中断服务程序的入口地址
3. 关中断

## 组合逻辑设计

设计步骤：
1. 列出操作时间表
2. 写出微操作命令的最简表达式
3. 画出逻辑图



## 组合逻辑设计

设计步骤：
1. 列出操作时间表

非访存指令

| 工作周期标记 | 节拍 | 状态条件 | 微操作命令信号 | CLA | COM | SHR | CSL | STP | ADD | STA | LDA | JMP | BAN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FE 取指 | $T_0$ | | PC ⟶ MAR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | 1 ⟶ R | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | $T_1$ | | M(MAR) ⟶ MDR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | ( PC ) +1 ⟶ PC | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | $T_2$ | | MDR ⟶ IR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | OP( IR ) ⟶ ID | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | I | 1 ⟶ IND | | | | | | 1 | 1 | 1 | 1 | 1 |
| | | $\overline{I}$ | 1 ⟶ EX | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

间址特征



## 组合逻辑设计

设计步骤：
1. 列出操作时间表

非访存指令

| 工作周期标记 | 节拍 | 状态条件 | 微操作命令信号 | CLA | COM | SHR | CSL | STP | ADD | STA | LDA | JMP | BAN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IND 间址 | $T_0$ | | Ad (IR) ⟶ MAR | | | | | | 1 | 1 | 1 | 1 | 1 |
| | | | 1 ⟶ R | | | | | | 1 | 1 | 1 | 1 | 1 |
| | $T_1$ | | M(MAR) ⟶ MDR | | | | | | 1 | 1 | 1 | 1 | 1 |
| | $T_2$ | | MDR ⟶ Ad (IR) | | | | | | 1 | 1 | 1 | 1 | 1 |
| | | $\overline{IND}$ | 1 ⟶ EX | | | | | | 1 | 1 | 1 | 1 | 1 |

间址周期标志



## 组合逻辑设计

设计步骤：
1. 列出操作时间表

2. 写出微操作命令的最简表达式

| 工作周期标记 | 节拍 | 状态条件 | 微操作命令信号 | CLA | COM | ADD | STA | LDA | JMP | BAN |
|---|---|---|---|---|---|---|---|---|---|---|
| EX 执行 | $T_0$ | | Ad (IR) ⟶ MAR | | | 1 | 1 | 1 | | |
| | | | 1 ⟶ R | | | 1 | | 1 | | |
| | | | 1 ⟶ W | | | | 1 | | | |
| | $T_1$ | | M(MAR) ⟶ MDR | | | 1 | | 1 | | |
| | | | AC ⟶ MDR | | | | 1 | | | |
| | $T_2$ | | (AC)+(MDR)→AC | | | 1 | | | | |
| | | | MDR ⟶ M(MAR) | | | | 1 | | | |
| | | | MDR ⟶ AC | | | | | 1 | | |
| | | | 0 ⟶ AC | 1 | | | | | | |
| | | | $\overline{AC}$ ⟶ AC | | 1 | | | | | |
| | | | Ad(IR) ⟶ PC | | | | | | 1 | |
| | | $A_0$ | Ad(IR) ⟶ PC | | | | | | | 1 |

## 微操作信号综合

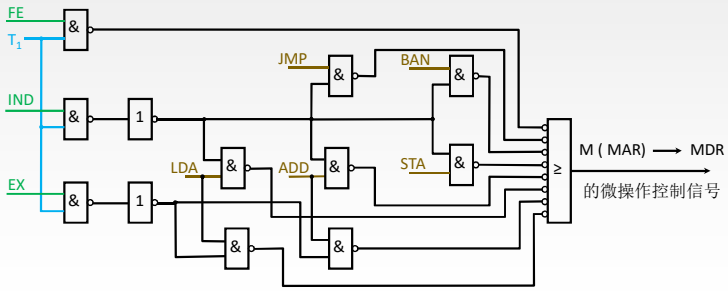
| 工作周期标记 | 节拍 | 状态条件 | 微操作命令信号 | CLA | COM | SHR | CSL | STP | ADD | STA | LDA | JMP | BAN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FE 取指 | $T_0$ | | PC ⟶ MAR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | 1 ⟶ R | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | $T_1$ | | M(MAR) ⟶ MDR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| IND 间址 | $T_1$ | | M(MAR) ⟶ MDR | | | | | | 1 | 1 | 1 | 1 | 1 |
| EX 执行 | $T_1$ | | 1 ⟶ W | | | | | | | 1 | | | |
| | | | M(MAR) ⟶ MDR | | | | | | 1 | | 1 | | |

M（MAR）→MDR微操作命令的逻辑表达式：
$FE \cdot T_1 + IND \cdot T_1(ADD+STA+LDA+JMP+BAN) + EX \cdot T_1(ADD+LDA)$
$= T_1\{FE+IND(ADD+STA+LDA+JMP+BAN)+EX(ADD+LDA)\}$



## 画出逻辑图

M（MAR）→MDR微操作命令的逻辑表达式：
$FE \cdot T_1 + IND \cdot T_1(ADD+STA+LDA+JMP+BAN) + EX \cdot T_1(ADD+LDA)$
$= T_1\{FE+IND(ADD+STA+LDA+JMP+BAN)+EX(ADD+LDA)\}$



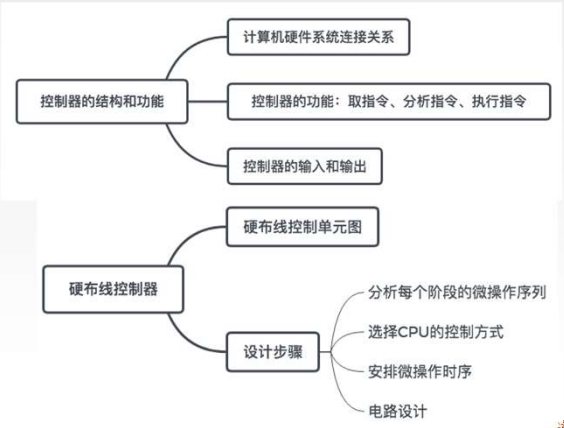



## 硬布线控制器小结

设计步骤：
1. 分析每个阶段的微操作序列
2. 选择CPU的控制方式
3. 安排微操作时序
4. 电路设计
   (1) 列出操作时间表
   (2)写出微操作命令的最简表达式
   (3)画出逻辑图



## 本节回顾



本节内容

# 中央处理器

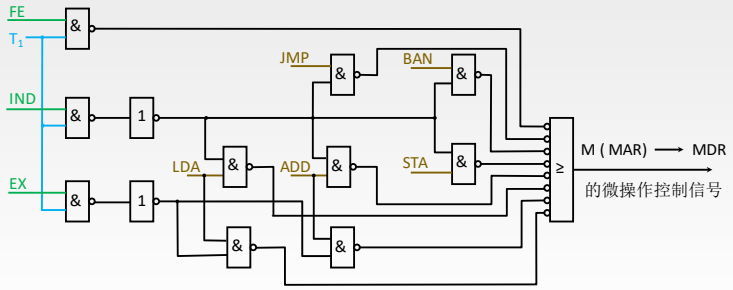
## 控制器的功能和工作原理
## 微程序



## 本章总览




## 控制器的设计思路

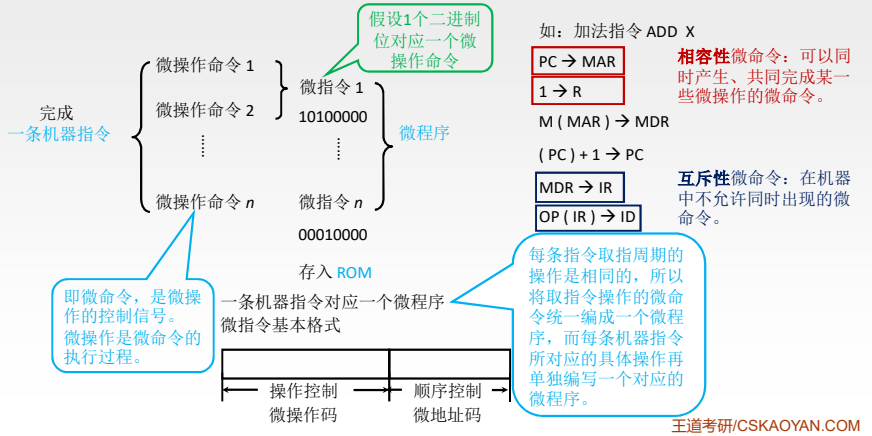



硬布线工作原理：**微操作控制信号**由组合逻辑电路根据当前的指令码、状态和时序，即时产生

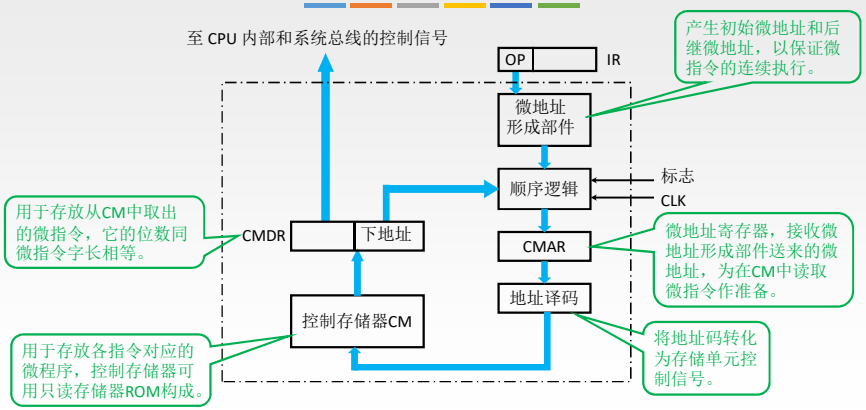
微程序工作原理：事先把**微操作控制信号**存储在一个专门的存储器(控制存储器)中，
将每一条机器指令编写成一个微程序，这些微程序可以存到一个控制存储器中，用寻址用户程序机器指令的办法来寻址每个微程序中的微指令。



## 微程序的基本思想

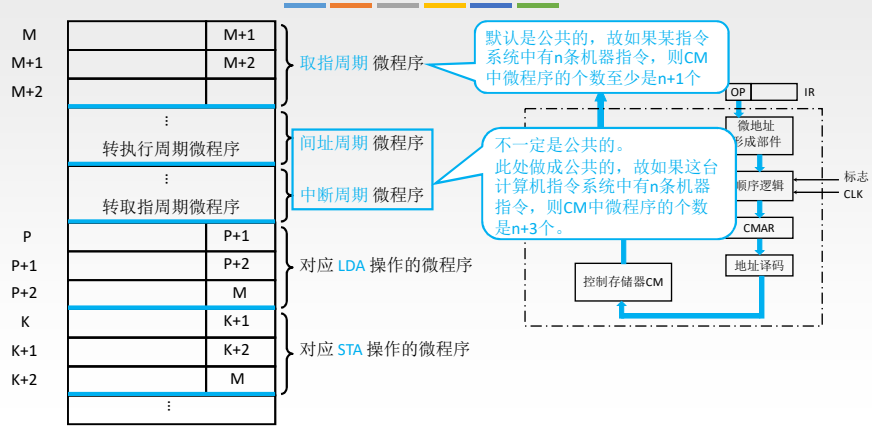


## 微程序控制器的基本结构






## 控制存储器






## 微指令的格式

1. 水平型微指令　一次能定义并执行多个并行操作。

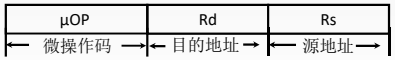
基本格式　如何表示一系列控制信号？


优点：微程序短，执行速度快；

缺点：微指令长，编写微程序较麻烦。

2. 垂直型微指令　**类似机器指令操作码**的方式，由微操作码字段规定微指令的功能。

基本格式




优点：微指令短、简单、规整，便于编写微程序；

缺点：微程序长，执行速度慢，工作效率低。

3. 混合型微指令　在垂直型的基础上增加一些不太复杂的并行操作。

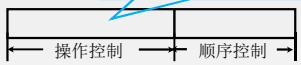
微指令较短，仍便于编写；微程序也不长，执行速度加快。



## 微指令的编码方式

1. 水平型微指令　一次能定义并执行多个并行操作。

基本格式　如何表示一系列控制信号？




优点：微程序短，执行速度快；

缺点：微指令长，编写微程序较麻烦。

微指令的编码方式又称为微指令的控制方式，它是指如何对微指令的控制字段进行编码，以形成控制信号。编码的目标是在保证速度的情况下，尽量缩短微指令字长。

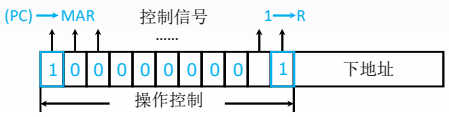
(1) 直接编码（直接控制）方式

在微指令的操作控制字段中，每一位代表一个微操作命令

某位为“1”表示该控制信号有效

优点：简单、直观，执行速度快，操作并行性好。

缺点：微指令字长过长，*n*个微命令就要求微指令的操作字段有*n*位，造成控存容量极大。



## 微指令的编码方式

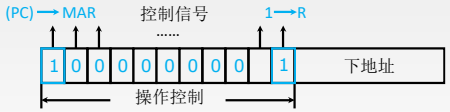
(1) 直接编码（直接控制）方式

在微指令的操作控制字段中，每一位代表一个微操作命令

某位为“1”表示该控制信号有效

优点：简单、直观，执行速度快，操作并行性好。

缺点：微指令字长过长，*n*个微命令就要求微指令的操作字段有*n*位，造成控存容量极大。




(2) 字段直接编码方式

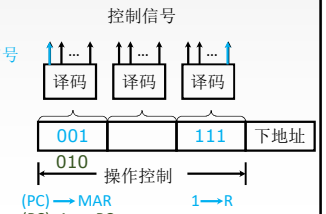
将微指令的控制字段分成若干“段”，每段经译码后发出控制信号

微命令字段分段的原则：

① 互斥性微命令分在同一段内，相容性微命令分在不同段内。

② 每个小段中包含的信息位不能太多，否则将增加译码线路的复杂性和译码时间。

③ 一般每个小段还要留出一个状态，表示本字段不发出任何微命令。因此，当某字段的长度为3位时，最多只能表示7个互斥的微命令，通常用000表示不操作。






## 微指令的编码方式

(2) 字段直接编码方式

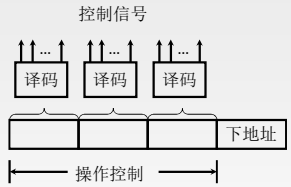
将微指令的控制字段分成若干“段”，每段经译码后发出控制信号

微命令字段分段的原则：

① 互斥性微命令分在同一段内，相容性微命令分在不同段内。

② 每个小段中包含的信息位不能太多，否则将增加译码线路的复杂性和译码时间。

③ 一般每个小段还要留出一个状态，表示本字段不发出任何微命令。因此，当某字段的长度为3位时，最多只能表示7个互斥的微命令，通常用000表示不操作。




**某计算机的控制器采用微程序控制方式，微指令中的操作控制字段采用字段直接编码法，共有33个微命令，构成5个互斥类，分别包含7、3、12、5和6个微命令，则操作控制字段至少有多少位？**

第1个互斥类有7个微命令，要留出1个状态表示不操作，所以需要表示8种不同的状态，故需要3个二进制位。

以此类推，后面4个互斥类各需要表示4、13、6、7种不同的状态，分别对应2、4、3、3个二进制位。

故操作控制字段的总位数为

3+2+4+3+3 = 15 位



## 微指令的编码方式

(2) 字段直接编码方式

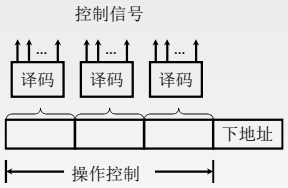
将微指令的控制字段分成若干“段”，每段经译码后发出控制信号

微命令字段分段的原则：

① 互斥性微命令分在同一段内，相容性微命令分在不同段内。

② 每个小段中包含的信息位不能太多，否则将增加译码线路的复杂性和译码时间。

③ 一般每个小段还要留出一个状态，表示本字段不发出任何微命令。因此，当某字段的长度为3位时，最多只能表示7个互斥的微命令，通常用000表示不操作。

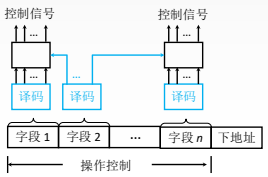



优点：可以缩短微指令字长 。

缺点：要通过译码电路后再发出微命令，因此比直接编码方式慢 。

(3) 字段间接编码方式

一个字段的某些微命令需由另一个字段中的某些微命令来解释，由于不是靠字段直接译码发出的微命令，故称为字段间接编码，又称隐式编码。




优点：可进一步缩短微指令字长。

缺点：削弱了微指令的并行控制能力，故通常作为字段直接编码方式的一种辅助手段。



## 微指令的地址形成方式

1. 微指令的 下地址字段 指出　微指令格式中设置一个下地址字段，由微指令的下地址字段直接指出后继微指令的地址，这种方式又称为**断定方式**。

2. 根据机器指令的 操作码 形成　当机器指令取至指令寄存器后，微指令的地址由操作码经微地址形成部件形成。

3. 增量计数器法　( CMAR ) + 1 → CMAR

4. 分支转移　转移方式：指明判别条件；转移地址：指明转移成功后的去向。

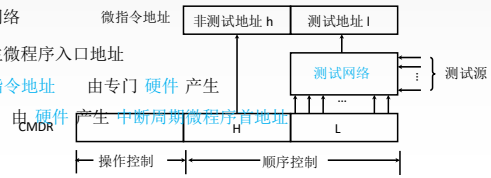
| 操作控制字段 | 转移方式 | 转移地址 |
|---|---|---|

5. 通过测试网络

6. 由硬件产生微程序入口地址

第一条微指令地址　由专门 硬件 产生

中断周期　由 硬件 产生 中断周期微程序首地址



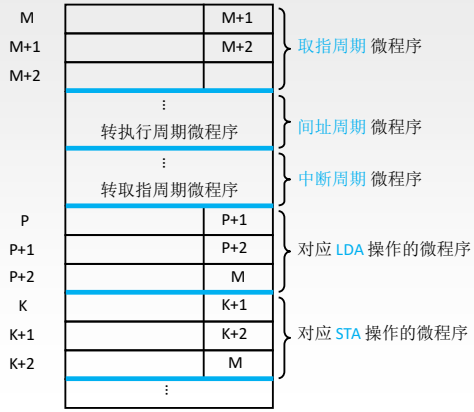
## 微指令的地址形成方式-断定方式

1. 微指令的 下地址字段 指出　微指令格式中设置一个下地址字段，由微指令的下地址字段直接指出后继微指令的地址，这种方式又称为**断定方式**。

某计算机采用微程序控制器，共有32条指令，公共的取指令微程序包含2条微指令，各指令对应的微程序平均由4条微指令组成，采用断定法（下地址字段法）确定下条微指令地址，则微指令中下地址字段的位数至少是多少位？



## 微指令的地址形成方式-断定方式




某计算机采用微程序控制器，共有32条指令，公共的取指令微程序包含2条微指令，各指令对应的微程序平均由4条微指令组成，采用断定法（下地址字段法）确定下条微指令地址，则微指令中下地址字段的位数至少是多少位？

总共需要存储多少条微指令？

$32\times4+2 = 130$条

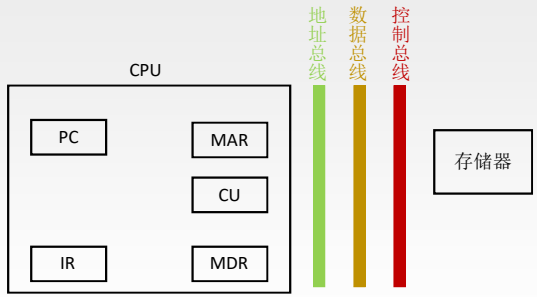
标注出130个不同的位置**至少**需要多少个二进制位？

$2^7 = 128$，$2^8 = 256$

下地址字段的位数至少是8位



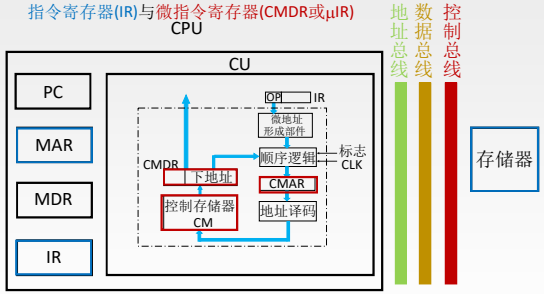
## 微程序控制的基本概念






## 微程序控制的基本概念

地址寄存器(MAR)与微地址寄存器(CMAR)
指令寄存器(IR)与微指令寄存器(CMDR或μIR)




**1. 微命令与微操作**

**微命令**是微操作的控制信号，
**微操作**是微命令的执行过程。

**2. 微指令与微周期**

**微指令**是若干微命令的集合。
**微周期**通常指从控制存储器中读取一条微指令并执行相应的微操作所需的时间。

**3. 主存储器与控制存储器**

**主存储器**用于存放程序和数据，在CPU外部，用RAM实现；
**控制存储器**（CM）用于存放微程序，在CPU内部，用ROM实现。

**4. 程序与微程序**

**程序**是指令的有序集合，用于完成特定的功能；
**微程序**是微指令的有序集合，一条指令的功能由一段微程序来实现。

## 微程序控制单元的设计

设计步骤：
1. 分析每个阶段的微操作序列
2. 写出对应机器指令的微操作命令及节拍安排
3. 确定微指令格式
4. 编写微指令码点

取指周期-硬布线控制器的节拍安排

$T_0$ PC → MAR
$T_0$ 1 → R
$T_1$ M ( MAR ) → MDR
$T_1$ ( PC ) + 1 → PC
$T_2$ MDR → IR
$T_2$ OP ( IR ) → ID

取指周期-微程序控制器的节拍安排

$T_0$ PC → MAR
$T_0$ 1 → R
$T_1$ M ( MAR ) → MDR
$T_1$ ( PC ) + 1 → PC
$T_2$ MDR → IR
$T_2$ OP ( IR ) → 微地址形成部件

3 条微指令

还需考虑 如何读出 这 3 条微指令



## 微程序控制单元的设计

取指周期-硬布线控制器的节拍安排

$T_0$ PC → MAR
$T_0$ 1 → R
$T_1$ M ( MAR ) → MDR
$T_1$ ( PC ) + 1 → PC
$T_2$ MDR → IR
$T_2$ OP ( IR ) → ID

取指周期-微程序控制器的节拍安排

$T_0$ PC → MAR
$T_0$ 1 → R
$T_1$ M ( MAR ) → MDR
$T_1$ ( PC ) + 1 → PC
$T_2$ MDR → IR
$T_2$ OP ( IR ) → 微地址形成部件

3 条微指令

还需考虑 如何读出 这 3 条微指令，
以及如何转入下一周期

Ad ( CMDR ) → CMAR

OP ( IR ) → 微地址形成部件 → CMAR



## 微程序控制单元的设计

取指周期-硬布线控制器的节拍安排

$T_0$ PC → MAR
$T_0$ 1 → R
$T_1$ M ( MAR ) → MDR
$T_1$ ( PC ) + 1 → PC
$T_2$ MDR → IR
$T_2$ OP ( IR ) → ID

取指周期-微程序控制器的节拍安排

$T_0$ PC → MAR
$T_0$ 1 → R
$T_1$ Ad ( CMDR ) → CMAR
$T_2$ M ( MAR ) → MDR
$T_2$ ( PC ) + 1 → PC
$T_3$ Ad ( CMDR ) → CMAR
$T_4$ MDR → IR
$T_4$ OP ( IR ) → 微地址形成部件
$T_5$ OP ( IR ) → CMAR

6 条微指令

还需考虑 如何读出 这 3 条微指令，
以及如何转入下一周期



## 微程序控制单元的设计

设计步骤：
1. 分析每个阶段的微操作序列
2. 写出对应机器指令的微操作命令及节拍安排
   (1) 写出每个周期所需要的微操作(参照硬布线)
   (2)补充微程序控制器特有的微操作：
      a. 取指周期：
         Ad ( CMDR ) → CMAR
         OP ( IR ) → CMAR
      b. 执行周期：
         Ad(CMDR) →CMAR
3. 确定微指令格式
4. 编写微指令码点

## 微程序控制单元的设计

设计步骤：

1. 分析每个阶段的微操作序列
2. 写出对应机器指令的微操作命令及节拍安排
   (1) 写出每个周期所需要的微操作(参照硬布线)
   (2)补充微程序控制器特有的微操作：
      a. 取指周期：
         Ad ( CMDR ) → CMAR
         OP ( IR ) → CMAR
      b. 执行周期：
         Ad(CMDR) →CMAR

3. 确定微指令格式

根据微操作个数决定采用何种编码方式，以确定微指令的操作控制字段的位数。
由微指令数确定微指令的顺序控制字段的位数。
最后按操作控制字段位数和顺序控制字段位数就可确定微指令字长。

4. 编写微指令码点



## 微程序控制单元的设计

设计步骤：

1. 分析每个阶段的微操作序列
2. 写出对应机器指令的微操作命令及节拍安排
   (1) 写出每个周期所需要的微操作(参照硬布线)
   (2)补充微程序控制器特有的微操作：
      a. 取指周期：
         Ad ( CMDR ) → CMAR
         OP ( IR ) → CMAR
      b. 执行周期：
         Ad(CMDR) →CMAR

3. 确定微指令格式

根据微操作个数决定采用何种编码方式，以确定微指令的操作控制字段的位数。
由微指令数确定微指令的顺序控制字段的位数。
最后按操作控制字段位数和顺序控制字段位数就可确定微指令字长。

4. 编写微指令码点

根据操作控制字段每一位代表的微操作命令，编写每一条微指令的码点。



## 微程序设计分类

1. 静态微程序设计和动态微程序设计

   静态　微程序无需改变，采用 ROM

   动态　通过 改变微指令 和 微程序 改变机器指令
   有利于仿真，采用 EPROM

2. 毫微程序设计

   毫微程序设计的基本概念

   微程序设计　用 微程序解释机器指令

   毫微程序设计　用 毫微程序解释微程序

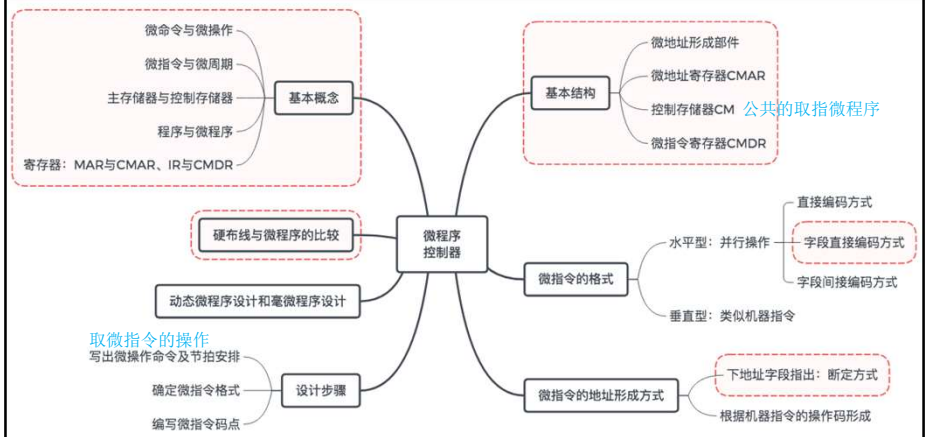
   毫微指令与微指令 的关系好比 微指令与机器指令 的关系



## 硬布线与微程序的比较

| 类别 / 对比项目 | 微程序控制器 | 硬布线控制器 |
|---|---|---|
| 工作原理 | 微操作控制信号以微程序的形式存放在控制存储器中，执行指令时读出即可 | 微操作控制信号由组合逻辑电路根据当前的指令码、状态和时序，即时产生 |
| 执行速度 | 慢 | 快 |
| 规整性 | 较规整 | 烦琐、不规整 |
| 应用场合 | CISC CPU | RISC CPU |
| 易扩充性 | 易扩充修改 | 困难 |

## 本节回顾





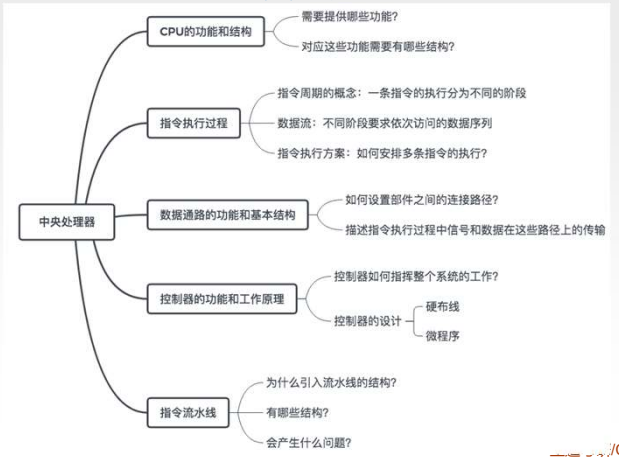

本节内容

# 中央处理器

## 指令流水线
## 基本概念
## 性能指标



## 本章总览






## 指令流水的定义

一条指令的执行过程可以分成多个阶段（或过程）。
根据计算机的不同，具体的分法也不同。

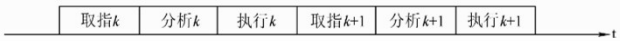
| 取指 | 分析 | 执行 |
|---|---|---|

特点：每个阶段用到的硬件不一样。

**取指**：根据PC内容访问主存储器，取出一条指令送到IR中。
**分析**：对指令操作码进行译码，按照给定的寻址方式和地址字段中的内容形成操作数的有效地址EA，并从有效地址EA中取出操作数。
**执行**：根据操作码字段，完成指令规定的功能，即把运算结果写到通用寄存器或主存中。

**设取指、分析、执行3个阶段的时间都相等，用 $t$表示 ，按以下几种执行方式分析n条指令的执行时间：**

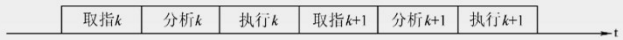
1. 顺序执行方式　　总耗时T = n×3t = 3nt




传统冯·诺依曼机采用顺序执行方式，又称串行执行方式。
优点：控制简单，硬件代价小。
缺点：执行指令的速度较慢，在任何时刻，处理机中只有一条指令在执行，各功能部件的利用率很低。

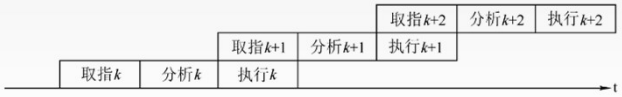
## 指令流水的定义

1. 顺序执行方式　总耗时T = n×3t = 3nt




传统冯·诺依曼机采用顺序执行方式，又称串行执行方式。
优点：控制简单，硬件代价小。
缺点：执行指令的速度较慢，在任何时刻，处理机中只有一条指令在执行，各功能部件的利用率很低。

2. 一次重叠执行方式　总耗时T = 3t + (n-1)×2t = (1+2n)t

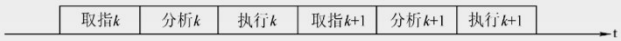



优点：程序的执行时间缩短了1/3，各功能部件的利用率明显提高。
缺点：需要付出硬件上较大开销的代价，控制过程也比顺序执行复杂了。



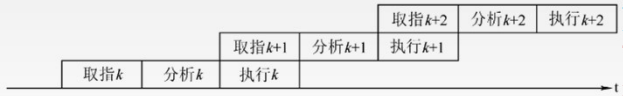
## 指令流水的定义

1. 顺序执行方式　总耗时T = n×3t = 3nt



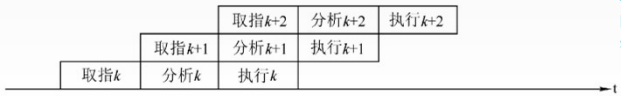

传统冯·诺依曼机采用顺序执行方式，又称串行执行方式。优点：控制简单，硬件代价小。
缺点：执行指令的速度较慢，在任何时刻，处理机中只有一条指令在执行，各功能部件的利用率很低。

2. 一次重叠执行方式　总耗时T = 3t + (n-1)×2t = (1+2n)t




优点：程序的执行时间缩短了1/3，各功能部件的利用率明显提高。
缺点：需要付出硬件上较大开销的代价，控制过程也比顺序执行复杂了。

3. 二次重叠执行方式　总耗时T = 3t + (n-1)×t = (2+n)t



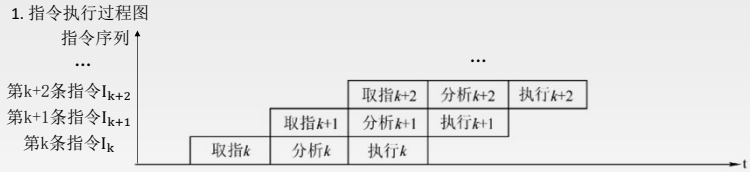

与顺序执行方式相比，指令的执行时间缩短近2/3。这是一种理想的指令执行方式，在正常情况下，处理机中同时有3条指令在执行。

注：也可以把每条指令的执行过程分成4个或5个阶段，分成5个阶段是比较常见的做法。



## 流水线的表示方法

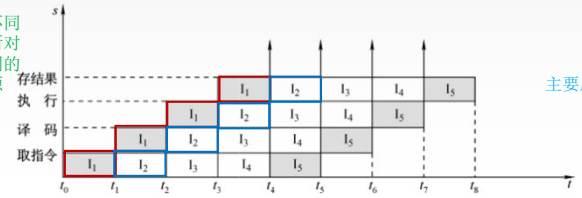
1. 指令执行过程图




主要用于分析指令执行过程以及影响流水线的因素(见下一个视频)

2. 时空图

空间：不同的阶段所对应的不同的硬件资源




主要用于分析流水线的性能

时间



## 流水线的性能指标

1. 吞吐率
2. 加速比
3. 效率

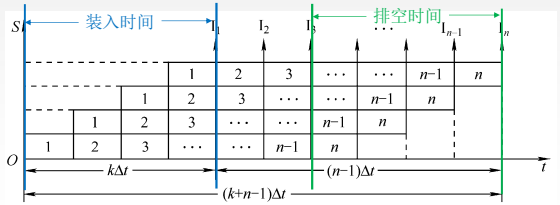
## 流水线的性能指标

1. 吞吐率　吞吐率是指在单位时间内流水线所完成的任务数量，或是输出结果的数量。

设任务数为$n$；处理完成$n$个任务所用的时间为$T_k$

则计算流水线吞吐率（TP）的最基本的公式为 $\mathrm{TP}=\dfrac{n}{T_k}$

**理想情况**下，流水线的时空图如下：　当连续输入的任务$n\to\infty$时，得最大吞吐率为$\mathrm{TP}_{max}=1/\Delta t$。




$$T_k=(k+n-1)\,\Delta t$$

流水线的实际吞吐率为

$$\mathrm{TP}=\frac{n}{(k+n-1)\Delta t}$$

一条指令的执行分为$k$个阶段，每个阶段耗时$\Delta t$，一般取$\Delta t$=一个时钟周期



## 流水线的性能指标

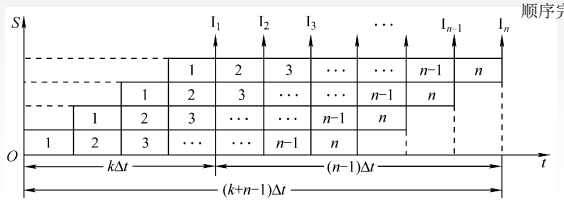
2. 加速比　完成同样一批任务，**不使用流水线所用的时间**与**使用流水线所用的时间**之比。

设$T_0$表示不使用流水线时的执行时间，即顺序执行所用的时间；$T_k$表示使用流水线时的执行时间

则计算流水线加速比（$S$）的基本公式为 $S=\dfrac{T_0}{T_k}$　当连续输入的任务$n\to\infty$时，最大加速比为$S_{max}=k$。

**理想情况**下，流水线的时空图如下：

单独完成一个任务耗时为$k\,\Delta t$，则顺序完成n个任务耗时$T_0=nk\,\Delta t$




$$T_k=(k+n-1)\,\Delta t$$

实际加速比为

$$S=\frac{kn\Delta t}{(k+n-1)\Delta t}=\frac{kn}{k+n-1}$$

一条指令的执行分为$k$个阶段，每个阶段耗时$\Delta t$，一般取$\Delta t$=一个时钟周期

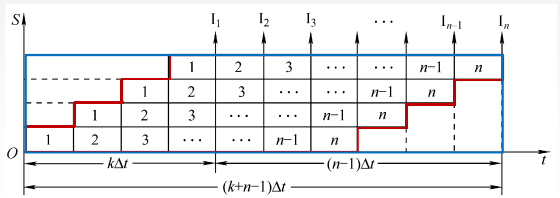


## 流水线的性能指标

3. 效率　流水线的设备利用率称为流水线的效率。
在时空图上，流水线的效率定义为**完成$n$个任务占用的时空区有效面积**与**$n$个任务所用的时间与$k$个流水段所围成的时空区总面积**之比。

则流水线效率（$E$）的一般公式为 $E=\dfrac{n\text{个任务占用}k\text{时空区有效面积}}{n\text{个任务所用的时间与}k\text{个流水段所围成的时空区总面积}}=\dfrac{T_0}{kT_k}$

**理想情况**下，流水线的时空图如下：

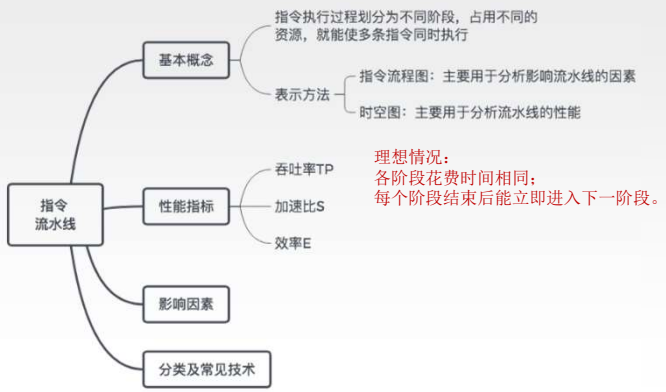



当连续输入的任务$n\to\infty$时，最高效率为$E_{max}=1$。

一条指令的执行分为$k$个阶段，每个阶段耗时$\Delta t$，一般取$\Delta t$=一个时钟周期



## 本节回顾



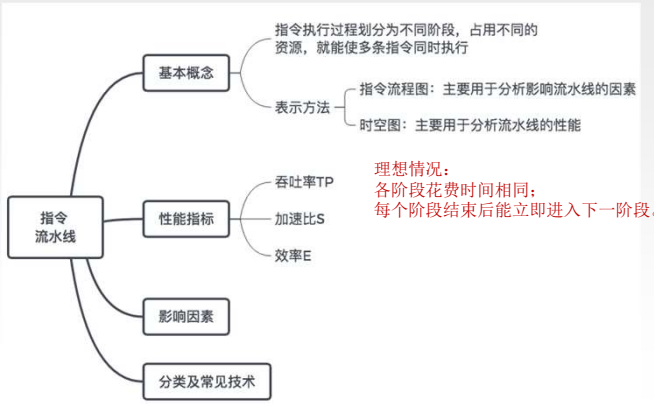
本节内容

# 中央处理器

## 指令流水线
## 影响因素
## 分类



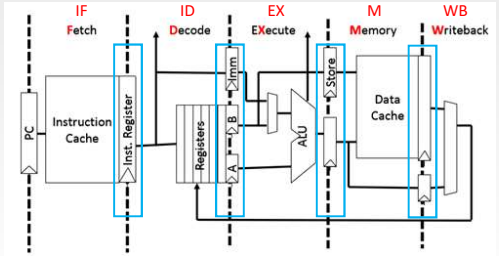
## 本节总览




理想情况：
各阶段花费时间相同；
每个阶段结束后能立即进入下一阶段。



## 机器周期的设置




流水线**每一个功能段部件后面**都要有一个**缓冲寄存器**，或称为**锁存器**，其作用是**保存本流水段的执行结果**，提供给下一流水段使用。

各部件实际耗时：　100ns　80ns　70ns　50ns　50ns

为方便流水线的设计，将每个阶段的耗时取成一样，以最长耗时为准。
即此处应将机器周期设置为100ns。



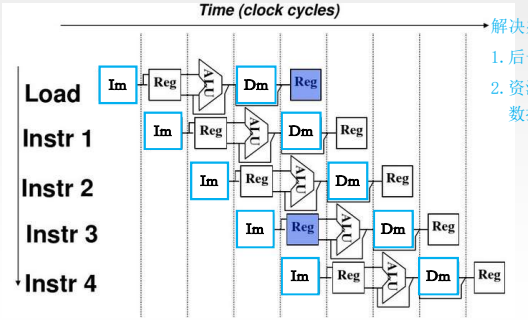
## 影响流水线的因素

1. **结构相关（资源冲突）**
2. **数据相关（数据冲突）**
3. **控制相关（控制冲突）**

## 影响流水线的因素

**1. 结构相关（资源冲突）**

由于多条指令在同一时刻争用同一资源而形成的冲突称为结构相关。




解决办法：

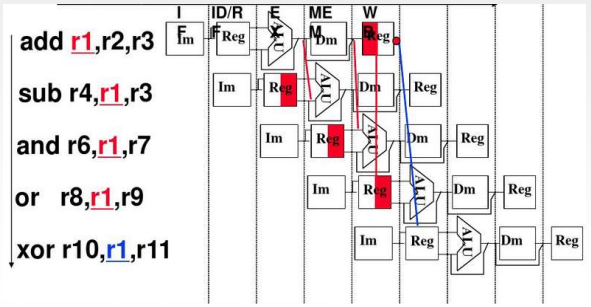
1. 后一相关指令暂停一周期
2. 资源重复配置：
   数据存储器+指令存储器



## 影响流水线的因素

**2. 数据相关（数据冲突）**

数据相关指在一个程序中，存在必须等前一条指令执行完才能执行后一条指令的情况，则这两条指令即为数据相关。




解决办法：

1. 把遇到数据相关的指令及其后续指令都暂停一至几个时钟周期，直到数据相关问题消失后再继续执行。可分为硬件阻塞(stall)和软件插入“NOP”两种方法。
2. 数据旁路技术。
3. 编译优化：通过编译器调整指令顺序来解决数据相关。



## 影响流水线的因素

**2. 数据相关（数据冲突）**

例题. 假设某指令流水线采用“按序发射，按序完成”方式，没有采用转发技术处理数据相关，并且同一寄存器的读和写操作不能在同一个时钟周期内进行。若高级语言程序中某赋值语句为x=a+b，x、a和b均为int型变量，它们的存储单元地址分别表示为[x]、[a]和[b]。该语句对应的指令序列及其在指令流中的执行过程如下图所示。

```
I1   LOAD   R1, [a]    ( [a] ) —> R1
I2   LOAD   R2, [b]    ( [b] ) —> R2
I3   ADD    R1, R2     ( R1 ) + ( R2 ) —> R2
I4   STORE  R2, [x]    ( R2 ) —> [x]
```

I3与I1和I2存在数据相关，I4和I3存在数据相关。

则这4条指令执行过程中I3的ID段和I4的IF段被阻塞的原因各是什么？

| | 时间单元 | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 指令 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| $I_1$ | IF | ID | EX | M | WB | | | | | | | | | |
| $I_2$ | | IF | ID | EX | M | WB | | | | | | | | |
| $I_3$ | | | IF | | | | ID | EX | M | WB | | | | |
| $I_4$ | | | | | | | IF | | | | ID | EX | M | WB |



## 影响流水线的因素

**2. 数据相关（数据冲突）**

数据的基本操作：读（R）、写（W）

冲突的基本类型：RAW、WAR、WAW

RAW

I1：ADD R5，R2，R4；(R2)+(R4) -> R5

I2：ADD R4，R5，R3；(R5)+(R3) -> R4

WAR

I1: STA M，R2；　(R2) -> M,M为主存单元

I2: ADD R2，R4，R5；　(R4)+(R5) ->R2

WRW

I1: MUL R3，R2，R1；　(R2)*(R1)->R3

I2: SUB R3，R4，R5；　(R4)-(R5)->R3

注：“按序发射，按序完成”时，只可能出现RAW相关。

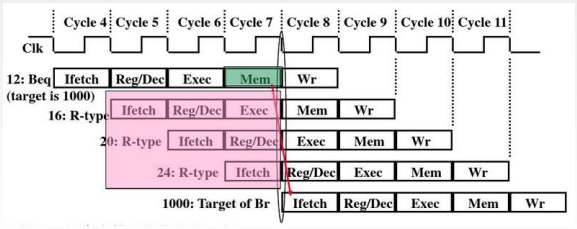
乱序发射，编写程序的时候希望I1在I2前完成，但优化手段导致I2在I1前发射。

存在多个功能部件时，后一条指令可能比前一条指令先完成。

## 影响流水线的因素

**3. 控制相关（控制冲突）**

当流水线遇到转移指令和其他改变PC值的指令而造成断流时，会引起控制相关。




解决办法：

1. 尽早判别转移是否发生，尽早生成转移目标地址
2. 预取转移成功和不成功两个控制流方向上的目标指令
3. 加快和提前形成条件码
4. 提高转移方向的猜准率



## 影响流水线的因素




## 流水线的分类

**1．部件功能级、处理机级和处理机间级流水线**

根据**流水线使用的级别**的不同，流水线可分为部件功能级流水线、处理机级流水线和处理机间流水线。

**部件功能级流水**就是将复杂的算术逻辑运算组成流水线工作方式。例如，可将浮点加法操作分成求阶差、对阶、尾数相加以及结果规格化等4个子过程。

**处理机级流水**是把一条指令解释过程分成多个子过程，如前面提到的取指、译码、执行、访存及写回5个子过程。

**处理机间流水**是一种宏流水，其中每一个处理机完成某一专门任务，各个处理机所得到的结果需存放在与下一个处理机所共享的存储器中。

**2．单功能流水线和多功能流水线**

按**流水线可以完成的功能**，流水线可分为单功能流水线和多功能流水线。

**单功能流水线**指只能实现一种固定的专门功能的流水线；

**多功能流水线**指通过各段间的不同连接方式可以同时或不同时地实现多种功能的流水线。



## 流水线的分类

**3．动态流水线和静态流水线**

按**同一时间内各段之间的连接方式**，流水线可分为静态流水线和动态流水线。

**静态流水线**指在同一时间内，流水线的各段只能按同一种功能的连接方式工作。

**动态流水线**指在同一时间内，当某些段正在实现某种运算时，另一些段却正在进行另一种运算。这样对提高流水线的效率很有好处，但会使流水线控制变得很复杂。

**4．线性流水线和非线性流水线**

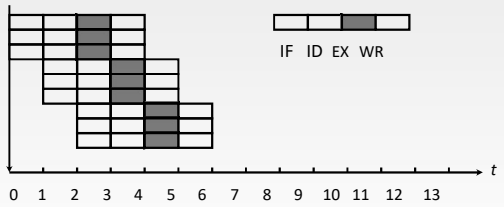
按流水线的**各个功能段之间是否有反馈信号**，流水线可分为线性流水线与非线性流水线。

**线性流水线**中，从输入到输出，每个功能段只允许经过一次，不存在反馈回路。

**非线性流水线**存在反馈回路，从输入到输出过程中，某些功能段将数次通过流水线，这种流水线适合进行线性递归的运算。

## 流水线的多发技术

**1. 超标量技术**



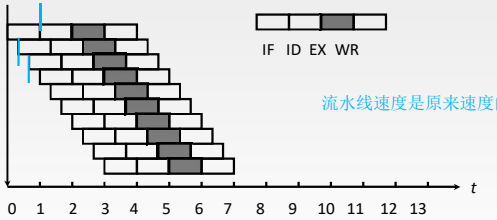

每个时钟周期内可 并发多条独立指令
要配置多个功能部件

不能调整 指令的 执行顺序
通过编译优化技术，把可并行执行的指令搭配起来



## 流水线的多发技术

**2. 超流水技术**




流水线速度是原来速度的 3 倍

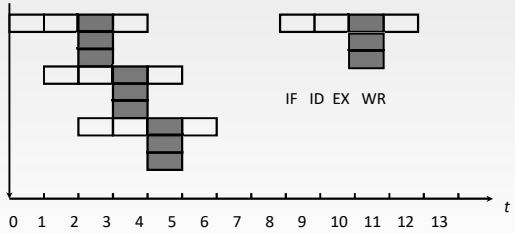
在 一个时钟周期 内 再分段 （ 3 段）
在一个时钟周期内 一个功能部件使用多次（ 3 次）

不能调整 指令的 执行顺序
靠编译程序解决优化问题



## 流水线的多发技术

**3. 超长指令字**




由 编译程序挖掘 出指令间 潜在 的 并行性，
将 多条 能 并行操作 的指令组合成 一条
具有 多个操作码字段 的 超长指令字（可达几百位）
采用 多个处理部件



## 本节回顾